

Multicore Real-Time Scheduling

Bjorn Andersson and Dionisio de Niz

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



Copyright 2015 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM-0002404

NASA related Roadmaps

C. L. Liu, “Scheduling algorithms for multiprocessors in a hard real-time environment,” JPL Space Programs Summary, pp. 37-60, 1969:



NASA related Roadmaps

NASA/TM-2013-217986/REV1, Flight Avionics Hardware Roadmap, Avionics Steering Committee, January 2014:

page (i):

“The ASC’s specific recommendations for near-term investments are: ... Rad Hard Multicore Processor”

page 34:

“CD07: Advanced COTS-Based Instrument Processor...As a follow on to CD3, this C&DH subsystem will utilize future generations of COTS devices.”

Steering Committee for NASA Technology Roadmaps; National Research Council of the National Academies, “NASA Space Technology Roadmaps and Priorities: Restoring NASA's Technological Edge and Paving the Way for a New Era in Space”:

page S-7 and S-8 in section “TOP TECHNICAL CHALLENGES”:

“C9) Improved Flight Computers: Develop advanced flight-capable devices and system software for real-time flight computing with low power, radiation-hard and fault-tolerant hardware that can be applied to autonomous landing, rendezvous and surface hazard avoidance.





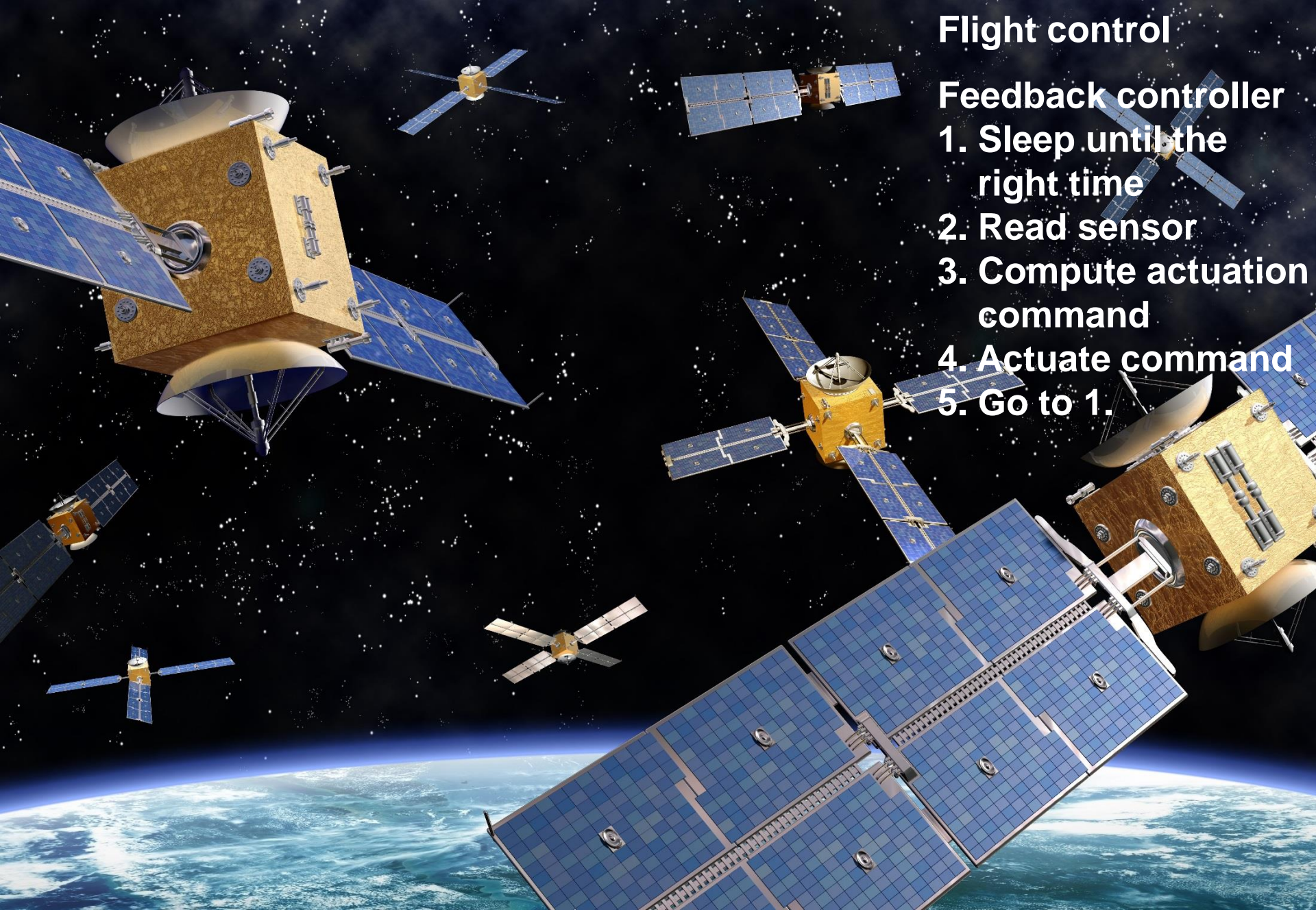


Flight control

Fly to the right position

**Avoid collisions
with space debris
other satellites**





Flight control

Feedback controller

1. Sleep until the right time
2. Read sensor
3. Compute actuation command
4. Actuate command
5. Go to 1.





Flight control

Feedback controller

1. Sleep until the right time
2. Read sensor
3. Compute actuation command
4. Actuate command
5. Go to 1.

The delay must
be at most
x milliseconds



Question: How to verify timing of software

Outline:

1. Challenges in verifying timing of software
2. Our track record
3. Specific challenges in verifying timing of software in autonomous systems



Challenges in verifying timing of software



Challenge 1: One processor, many threads

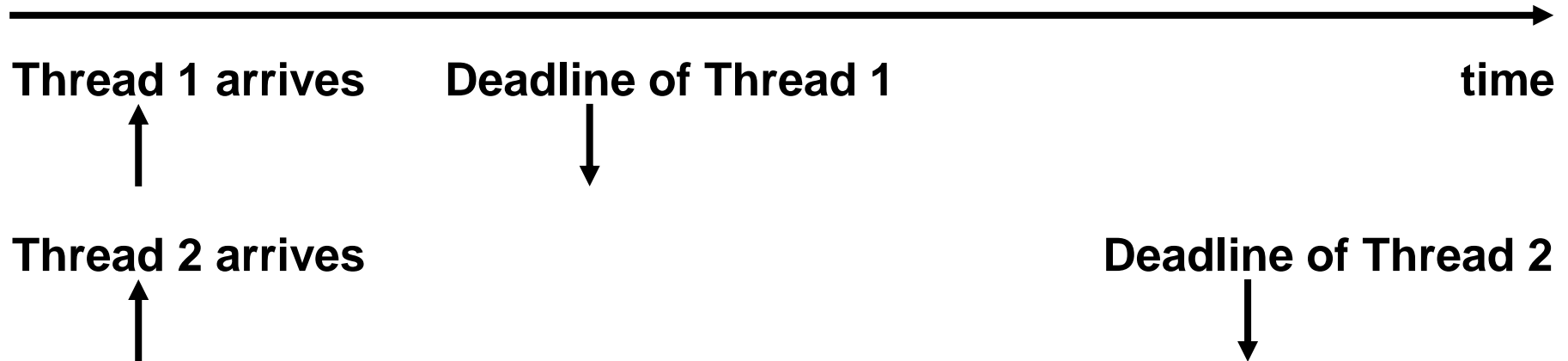
Thread 1

Thread 2



Challenge 1: One processor, many threads

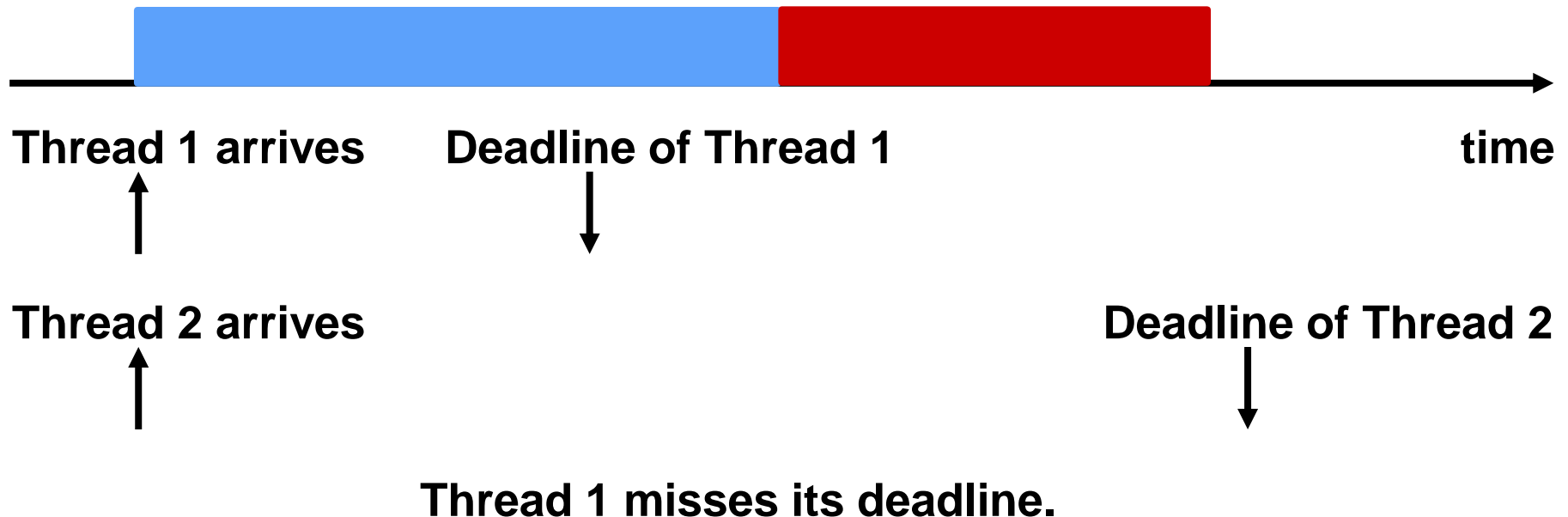
Thread 1 **Thread 2**



Challenge 1: One processor, many threads

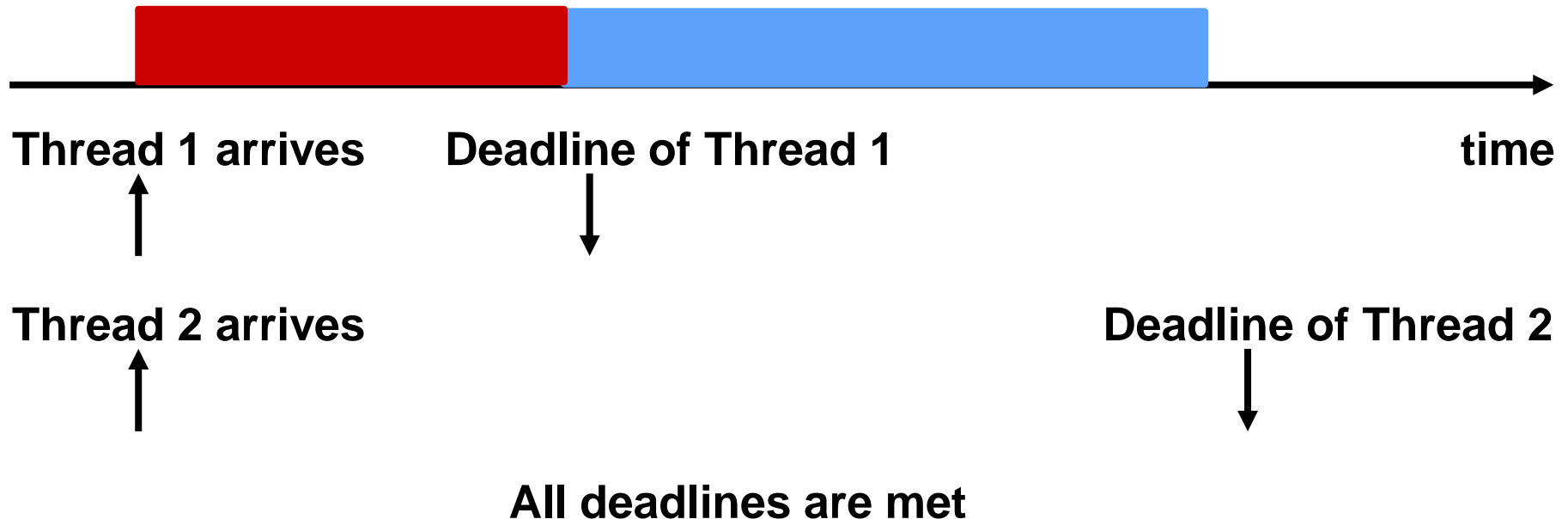
Thread 1

Thread 2



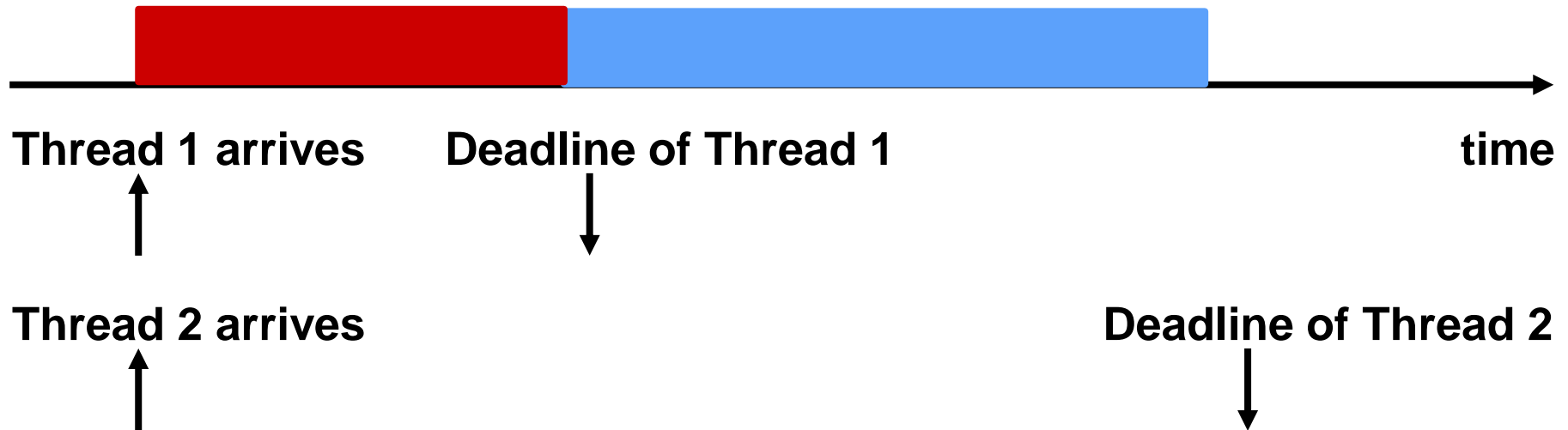
Challenge 1: One processor, many threads

Thread 1 **Thread 2**



Challenge 1: One processor, many threads

Thread 1 Thread 2



Good idea: priority of a thread is a function of its deadline.
Deadline-Monotonic (DM)

Challenge 2: Priority inversion, critical sections

Thread 1

Thread 2

Thread 3



Thread 1 and Thread 3 use critical section S

Assign priorities so a thread with short deadline has high priority (DM)



Challenge 2: Priority inversion, critical sections

Thread 1

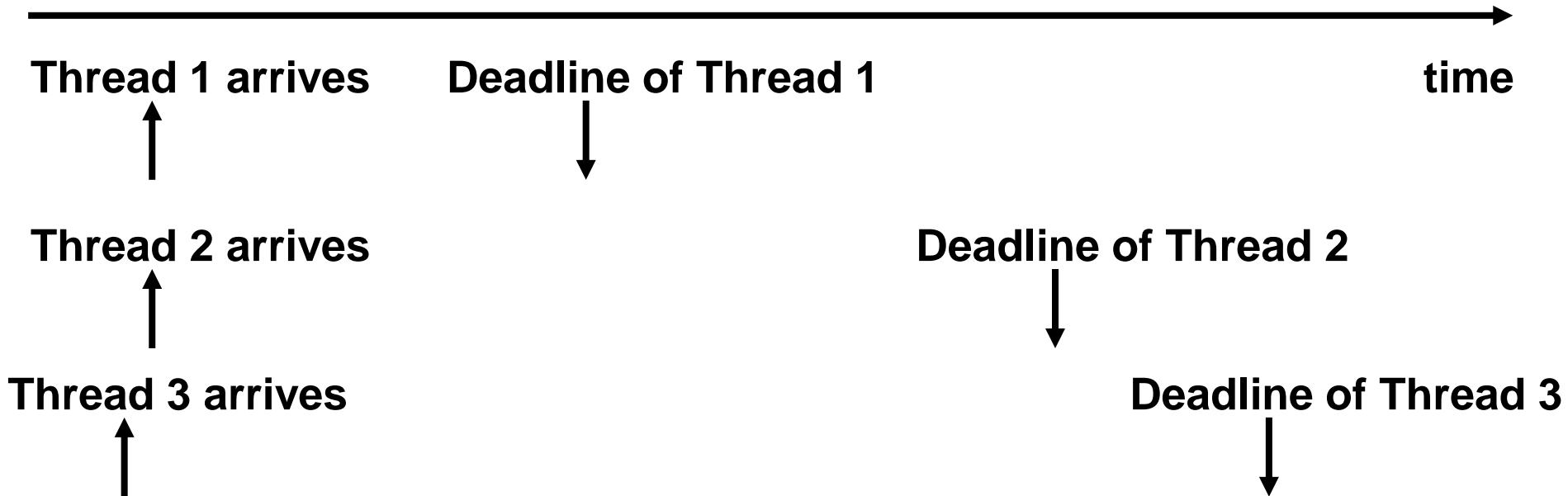
Thread 2

Thread 3



Thread 1 and Thread 3 use critical section S

Assign priorities so a thread with short deadline has high priority (DM)



Challenge 2: Priority inversion, critical sections

Thread 1

Thread 2

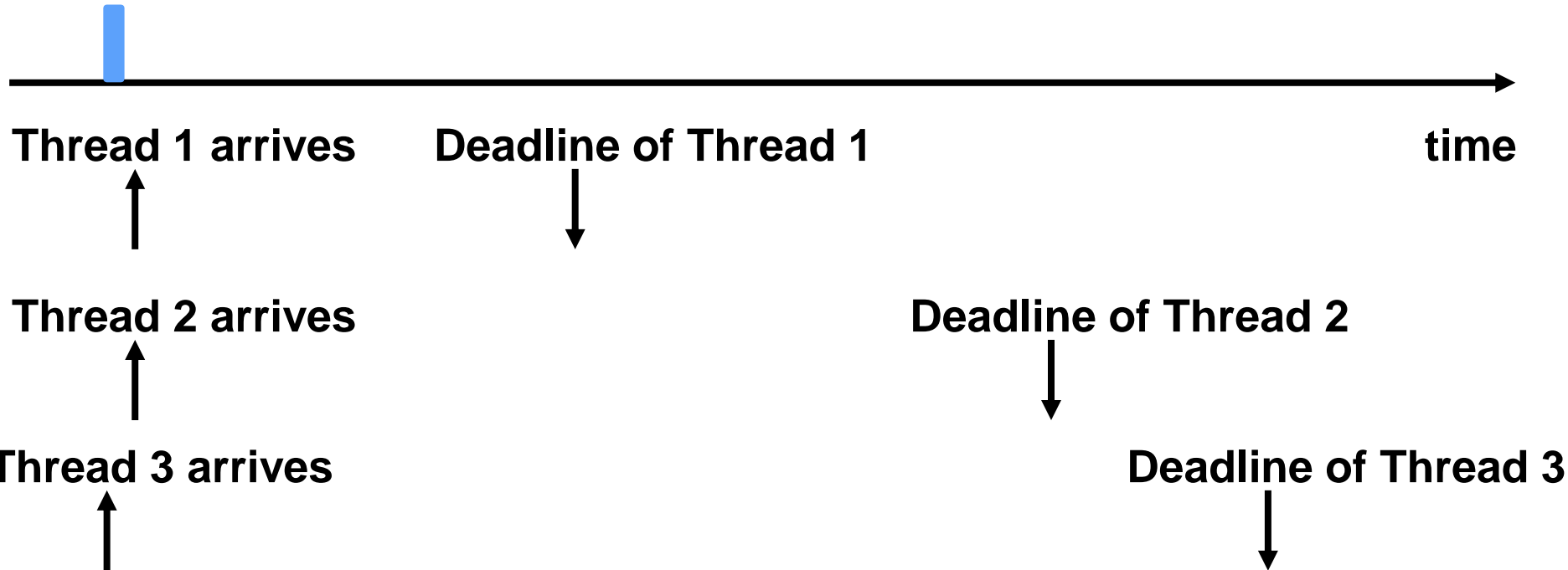
Thread 3



Thread 1 and Thread 3 use critical section S

Assign priorities so a thread with short deadline has high priority (DM)

Thread 3: Lock S



Challenge 2: Priority inversion, critical sections

Thread 1

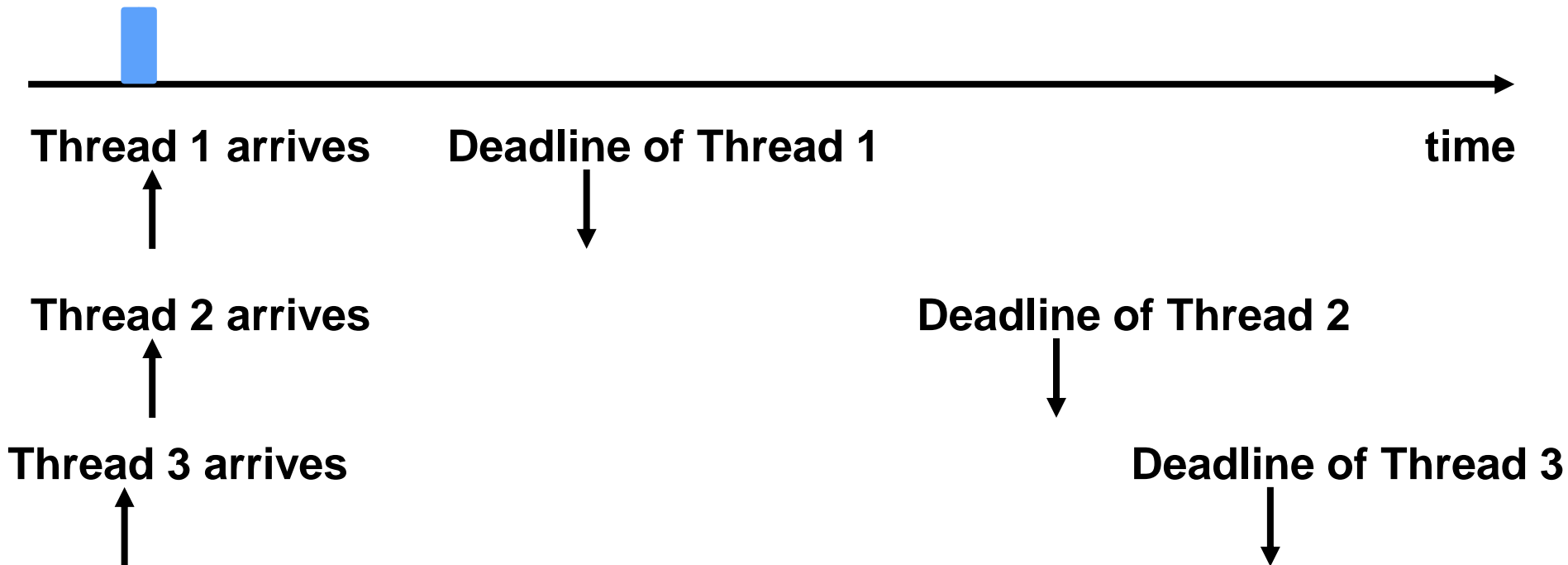
Thread 2

Thread 3



Thread 1 and Thread 3 use critical section S

Assign priorities so a thread with short deadline has high priority (DM)



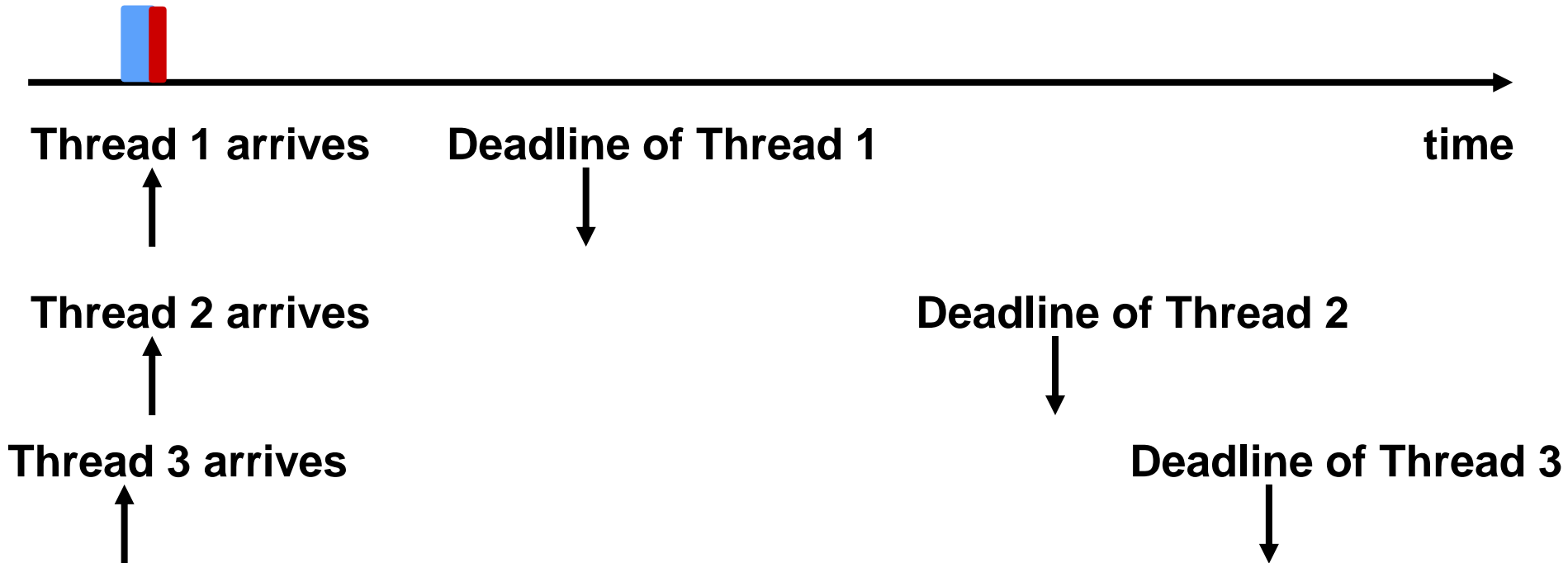
Challenge 2: Priority inversion, critical sections

Thread 1 **Thread 2** **Thread 3**



Thread 1 and Thread 3 use critical section S

Assign priorities so a thread with short deadline has high priority (DM)



Challenge 2: Priority inversion, critical sections

Thread 1

Thread 2

Thread 3



Thread 1 and Thread 3 use critical section S

Assign priorities so a thread with short deadline has high priority (DM)

Thread 1: Try to Lock S, failed, Thread 1 is blocked



Thread 1 arrives

Deadline of Thread 1

time

Thread 2 arrives

Deadline of Thread 2

Thread 3 arrives

Deadline of Thread 3



Challenge 2: Priority inversion, critical sections

Thread 1

Thread 2

Thread 3



Thread 2 executes

Thread 1 and Thread 3 use critical section S

Assign priorities so a thread with short deadline has high priority (DM)



Thread 1 arrives

Deadline of Thread 1

time

Thread 2 arrives

Deadline of Thread 2

Thread 3 arrives

Deadline of Thread 3



Challenge 2: Priority inversion, critical sections

Thread 1 **Thread 2** **Thread 3**



Thread 1 and Thread 3 use critical section S

Assign priorities so a thread with short deadline has high priority (DM)

Thread 2 has finished; Thread 3 executes



Thread 1 arrives

Deadline of Thread 1

time

Thread 2 arrives

Deadline of Thread 2

Thread 3 arrives

Deadline of Thread 3



Challenge 2: Priority inversion, critical sections

Thread 1

Thread 2

Thread 3



Thread 1 and Thread 3 use critical section S

Assign priorities so a thread with short deadline has high priority (DM)

Thread 3 has finished;
Thread 1 executes



Thread 1 arrives

Deadline of Thread 1

time

Thread 2 arrives

Deadline of Thread 2

Thread 3 arrives

Deadline of Thread 3



Challenge 2: Priority inversion, critical sections

Thread 1

Thread 2

Thread 3



Thread 1 and Thread 3 use critical section S

Assign priorities so a thread with short deadline has high priority (DM)

Thread 3 has finished;
Thread 1 executes



Thread 1 arrives

Deadline of Thread 1

time

Thread 2 arrives

Deadline of Thread 2

Thread 3 arrives

Deadline of Thread 3

Thread 1 misses its deadline.



Challenge 2: Priority inversion, critical sections

Thread 1

Thread 2

Thread 3



Thread 1 and Thread 3 use critical section S

Assign priorities so a thread with short deadline has high priority (DM)



Thread 1 misses its deadline.



Challenge 2: Priority inversion, critical sections

Thread 1

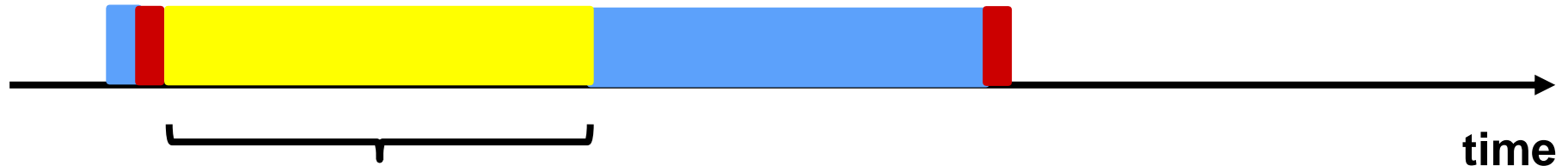
Thread 2

Thread 3



Thread 1 and Thread 3 use critical section S

Assign priorities so a thread with short deadline has high priority (DM)



Thread 1 waits for a lower priority thread with whom it does not share a critical section

This situation almost caused a mission failure of an autonomous system (see NASA Mars Pathfinder 1997).



Challenge 3: Memory interference in multicore processors

Thread 1

Thread 2

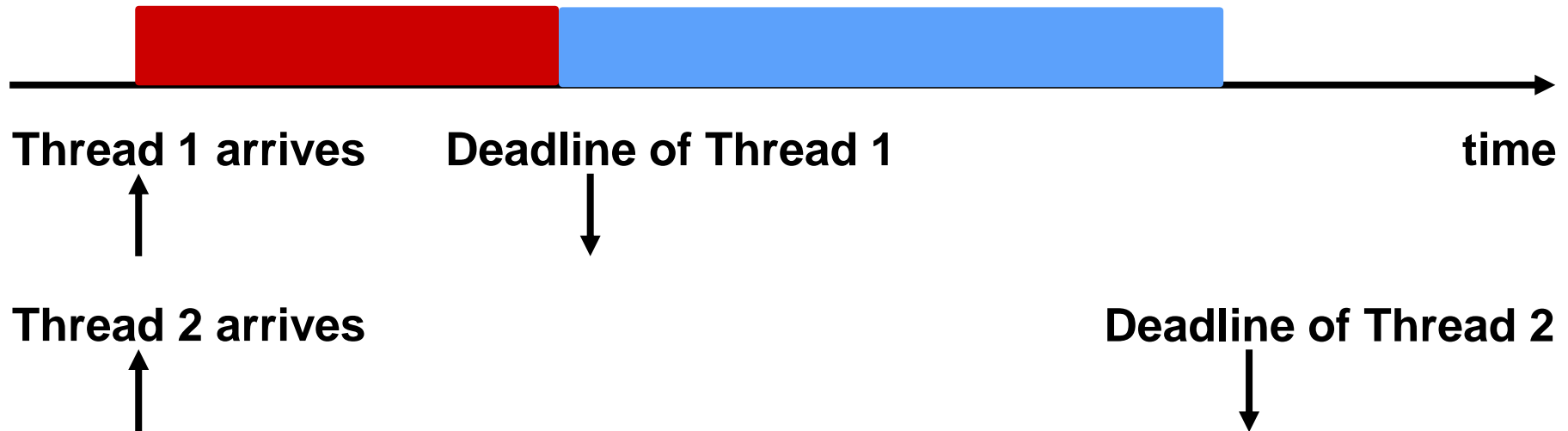


Let us consider a system with a single processor first.

Challenge 3: Memory interference in multicore processors

Thread 1

Thread 2



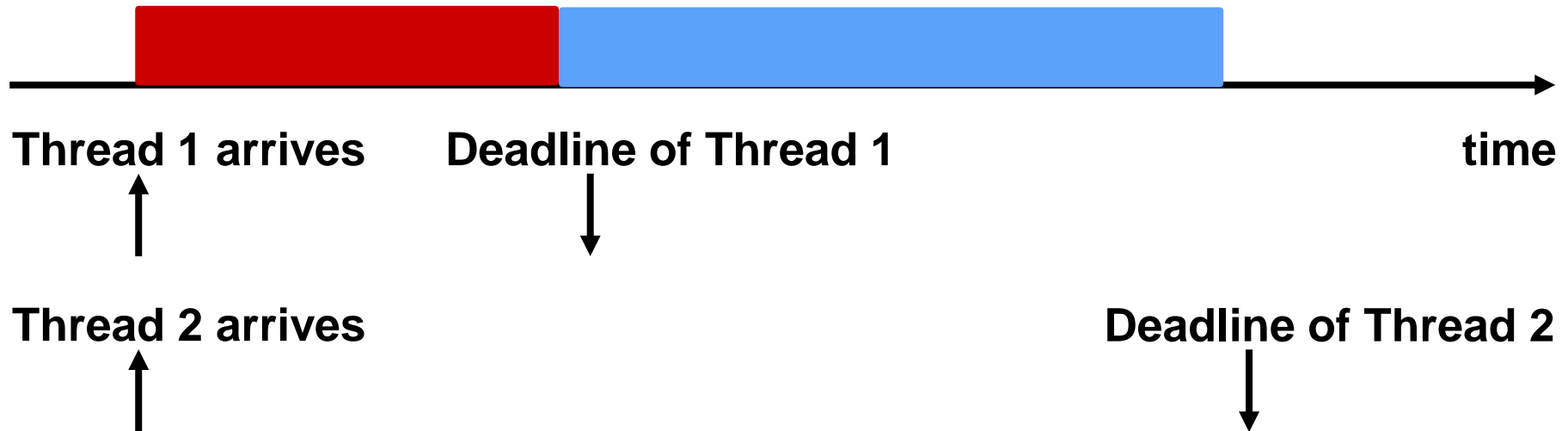
All deadlines are met



Challenge 3: Memory interference in multicore processors

Thread 1

Thread 2



Let us migrate this software to a multiprocessor with two processors.



Challenge 3: Memory interference in multicore processors

Thread 1

Thread 2



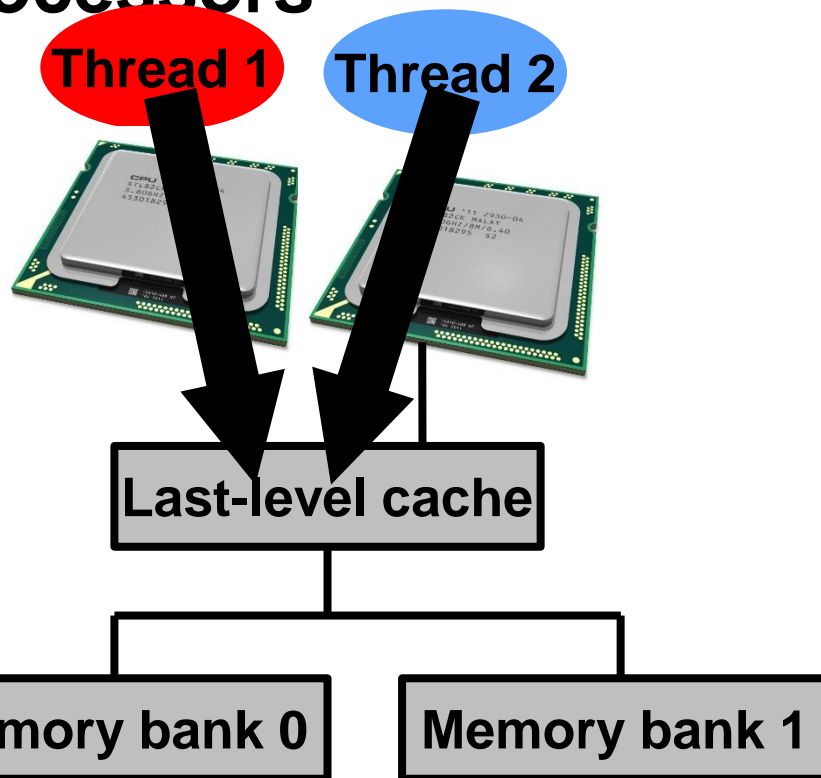
Last-level cache

Memory bank 0

Memory bank 1

Processors share memory bus
Processors share last-level cache

Challenge 3: Memory interference in multicore processors

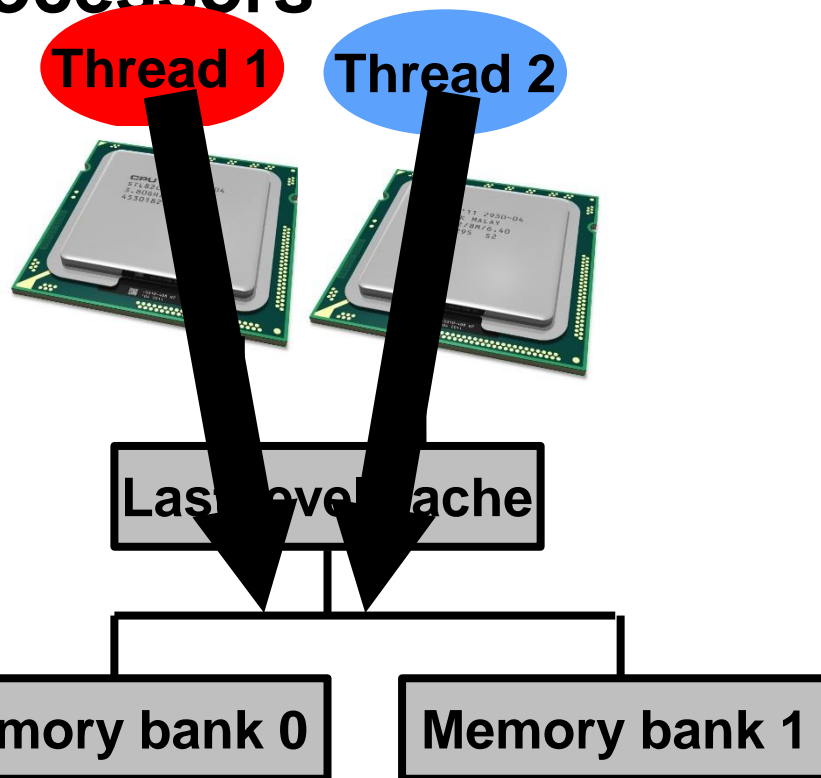


Processors share memory bus
Processors share last-level cache

Thread 1 can evict a cache block
that Thread 2 brought into the
last-level cache.

⇒ slowdown of execution

Challenge 3: Memory interference in multicore processors



Processors share memory bus
Processors share last-level cache

Thread 1 and Thread 2 may request the memory bus simultaneously but only one can be served at a time
⇒ slowdown of execution

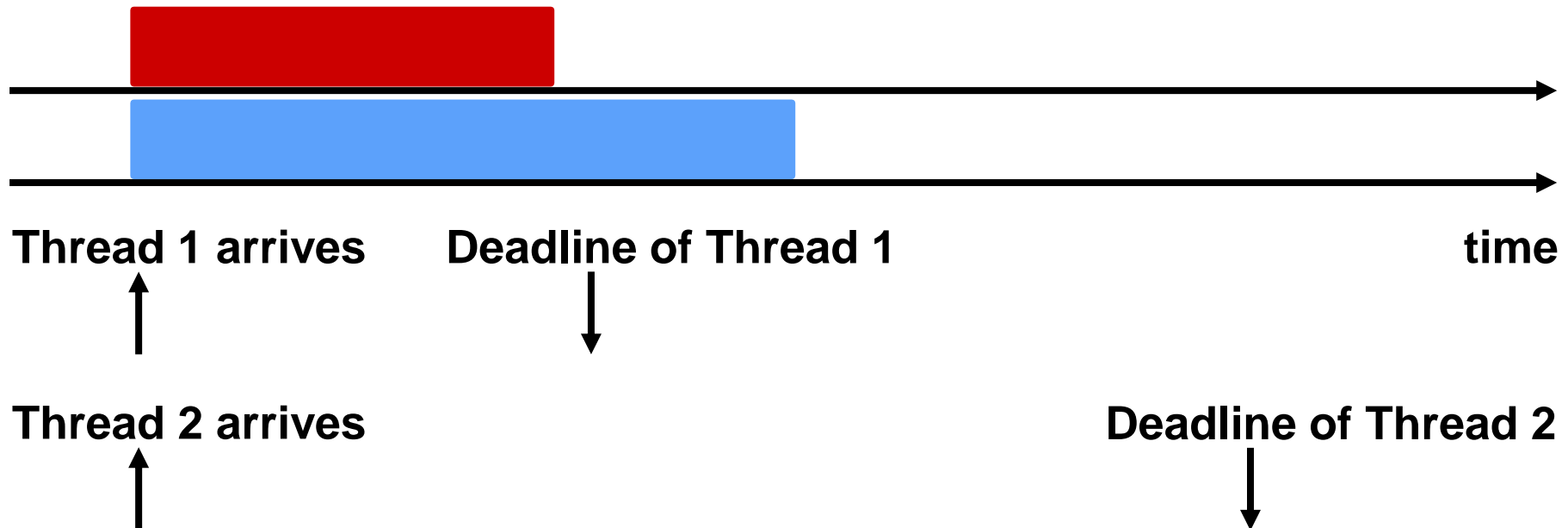
Challenge 3: Memory interference in multicore processors

Thread 1

Thread 2



Processors share memory bus
Processors share last-level cache



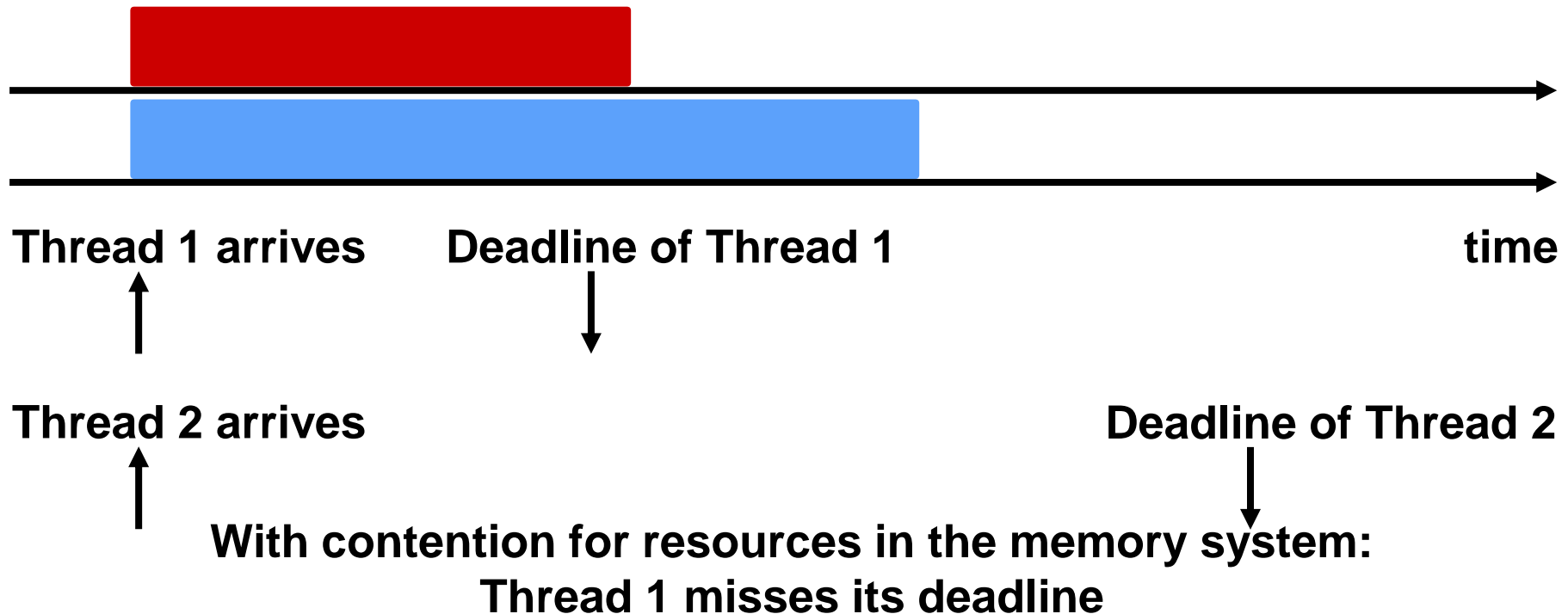
Assuming no memory contention: All deadlines are met



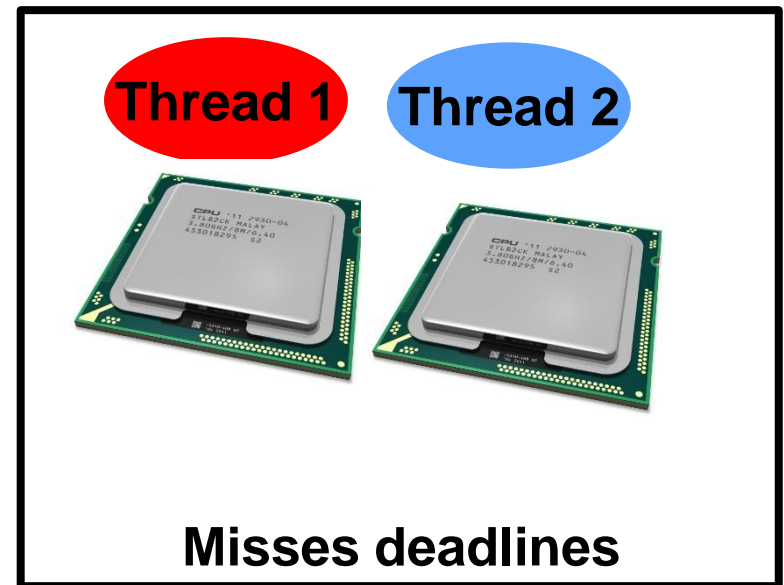
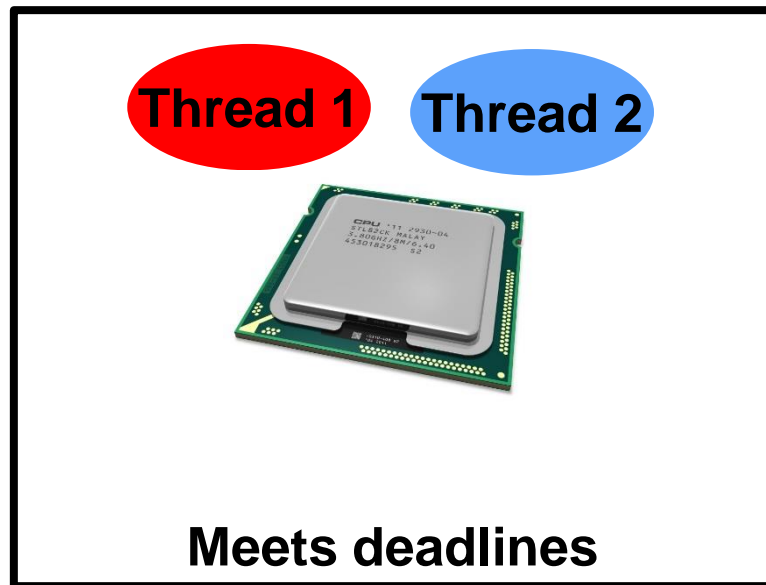
Challenge 3: Memory interference in multicore processors

Thread 1

Thread 2



Challenge 3: Memory interference in multicore processors

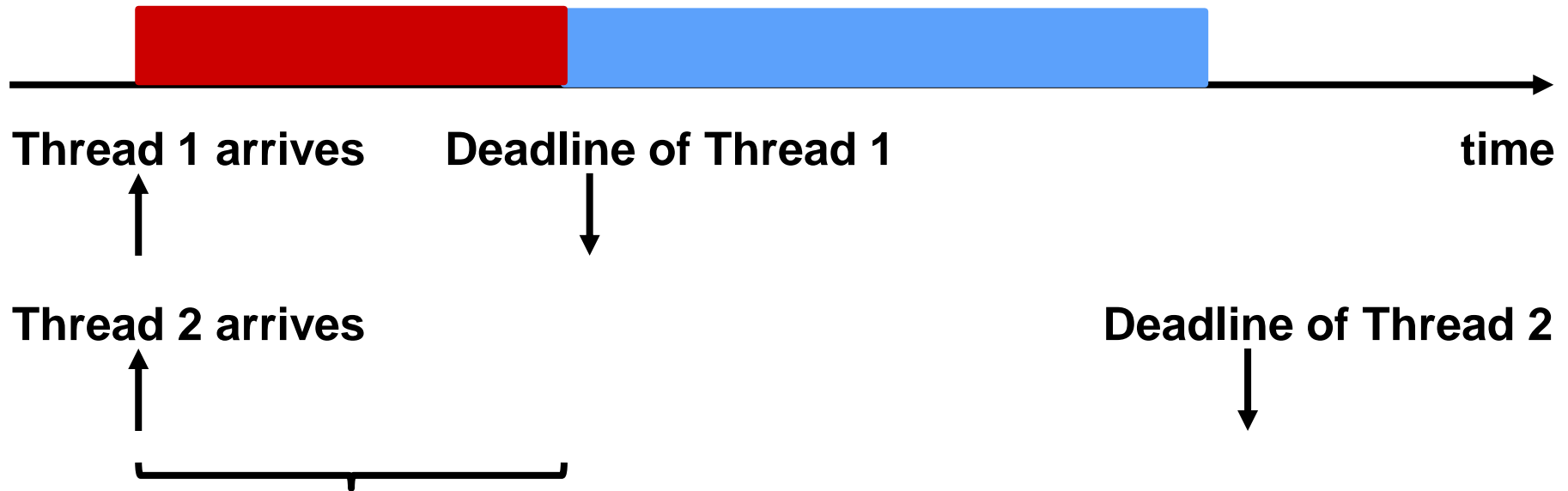


Upgrading a software system to multicore hardware
can cause a deadline miss.

Challenge 4: Execution overruns

Thread 1

Thread 2



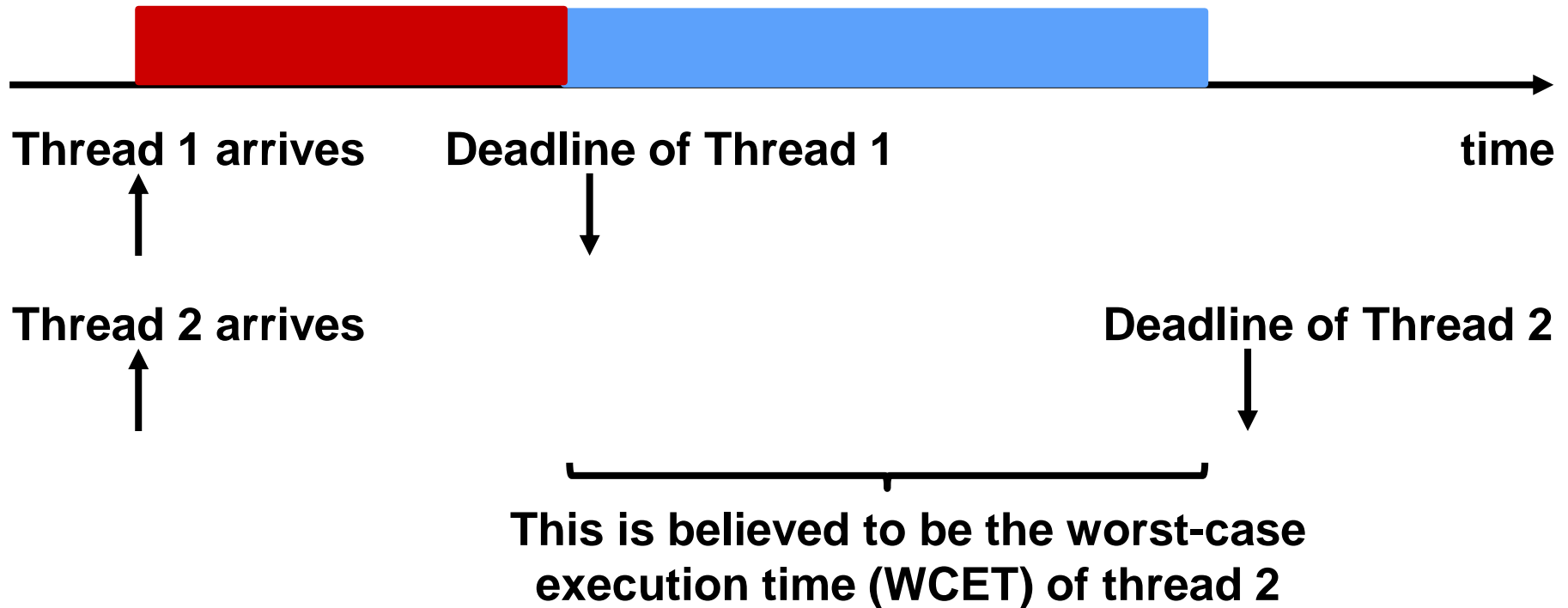
This is believed to be the worst-case execution time (WCET) of thread 1



Challenge 4: Execution overruns

Thread 1

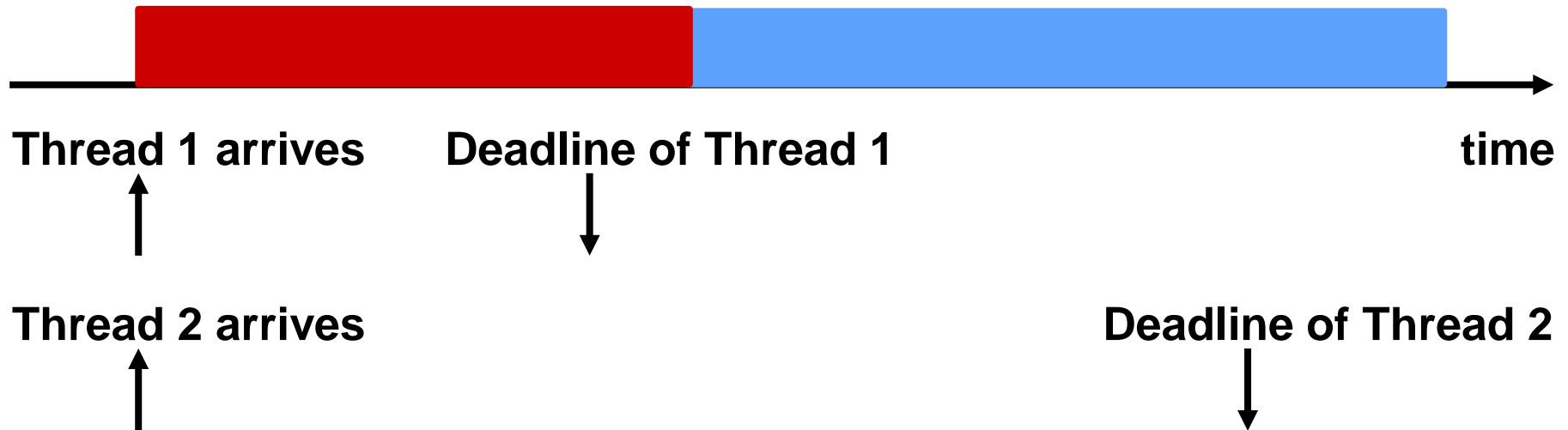
Thread 2



Challenge 4: Execution overruns

Thread 1

Thread 2



If a thread executes for longer than its believed worst-case execution time, then a deadline may be missed.



Challenge 5: Mode change

Thread 1

Thread 2



Challenge 5: Mode change

Thread 1

Thread 2



Thread 1

Thread 3



Autonomous system
is requested to adapt

time



Challenge 5: Mode change

Thread 1

Thread 2



Mode 1

Thread 1

Thread 3



Mode 2

Autonomous system
is requested to adapt

time

We need to prove that Thread 1 does not miss a deadline during the transition from Mode 1 to Mode 2.

Our track record



Challenge 1,2,3:

Previous work on single processor

RM, 0.69

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j$$

Priority ceiling protocol and
priority inheritance protocol

Our work on multiprocessor

RM-US(0.33), $0.33 * m$

$$R_i = C_i + \frac{1}{m} * \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil + 1 \right) * C_j$$

First analysis of priority
inheritance protocol for (global)
multiprocessor (RTSS'09)

First method for analyzing contention
on memory bus (RTSS-WIP'09)

First coordinated cache and bank
coloring (ICESS'13)

First method for analyzing contention
on memory bus considering bank
sharing (RTAS'14)



Challenge 4,5:

**First implementation of mixed-criticality scheduler in OS-kernel
VX Works, under evaluation by NASA**

**First locking protocol for mixed-criticality scheduling
(RTAS'11)**

**First mode change protocol and analysis for EDF
(OPODIS'08)**

**First mode change protocol with mode-independent tasks on a
multiprocessor
(ECRTS'11)**



Timing challenges specific to autonomous systems



Challenge 6: The execution time of a thread is highly variable.

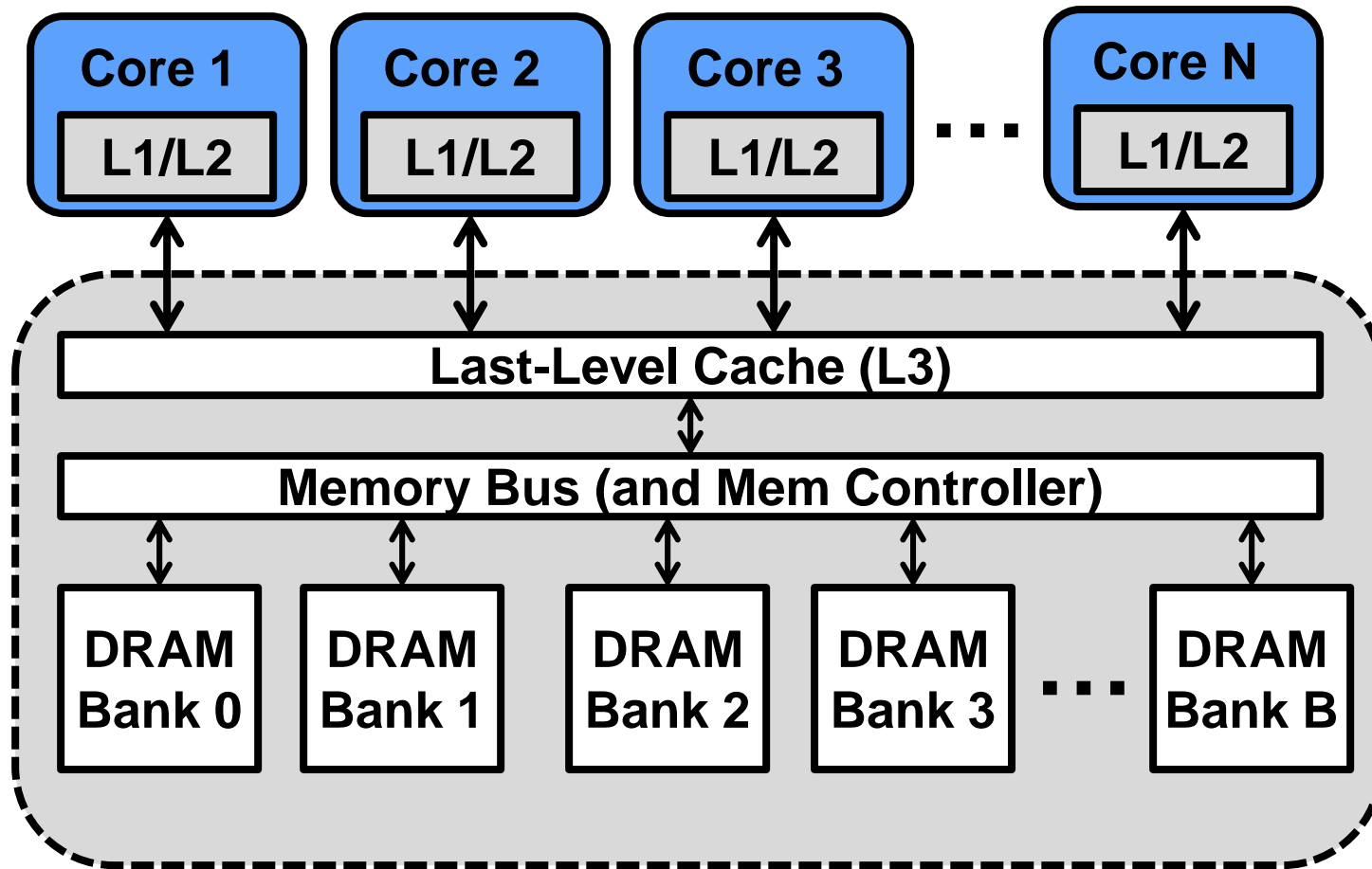
Challenge 7: A thread may not even terminate.

Challenge 8: The environment is unknown and hence the number of events that the software needs to process is not known before run-time.

Challenge 9: The execution of the software depends on the physical world and the physical world depends on the software.



Sharing of Multiple Hardware Resources



Need of Coordinated Protection

Need to constrain interference through each resource type

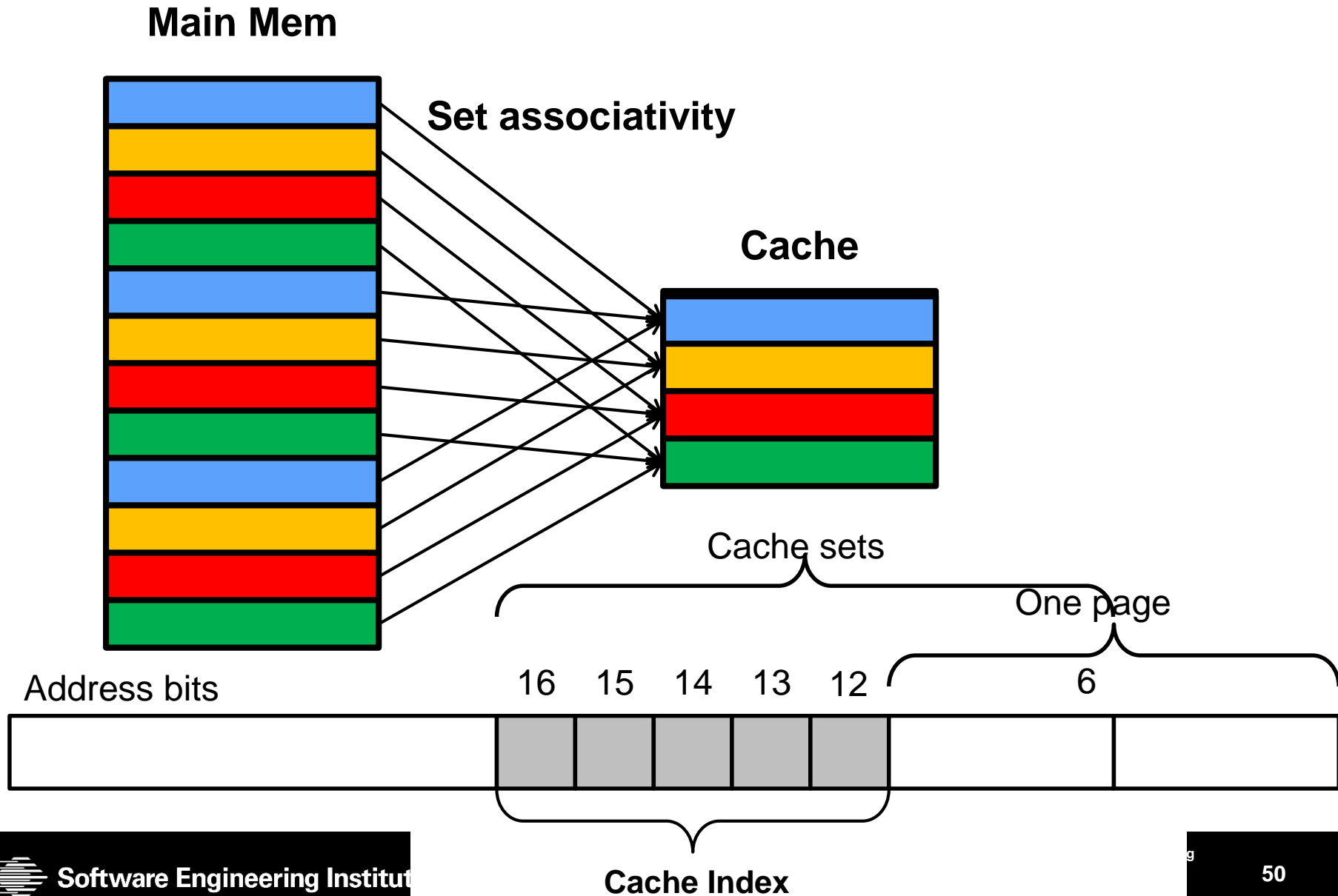
- CPU cycles
- Cache
- Memory Banks
- Memory Bus / inter-core network

Ensure no inconsistent configuration

- Configuration for one resource does not invalidate configuration of another

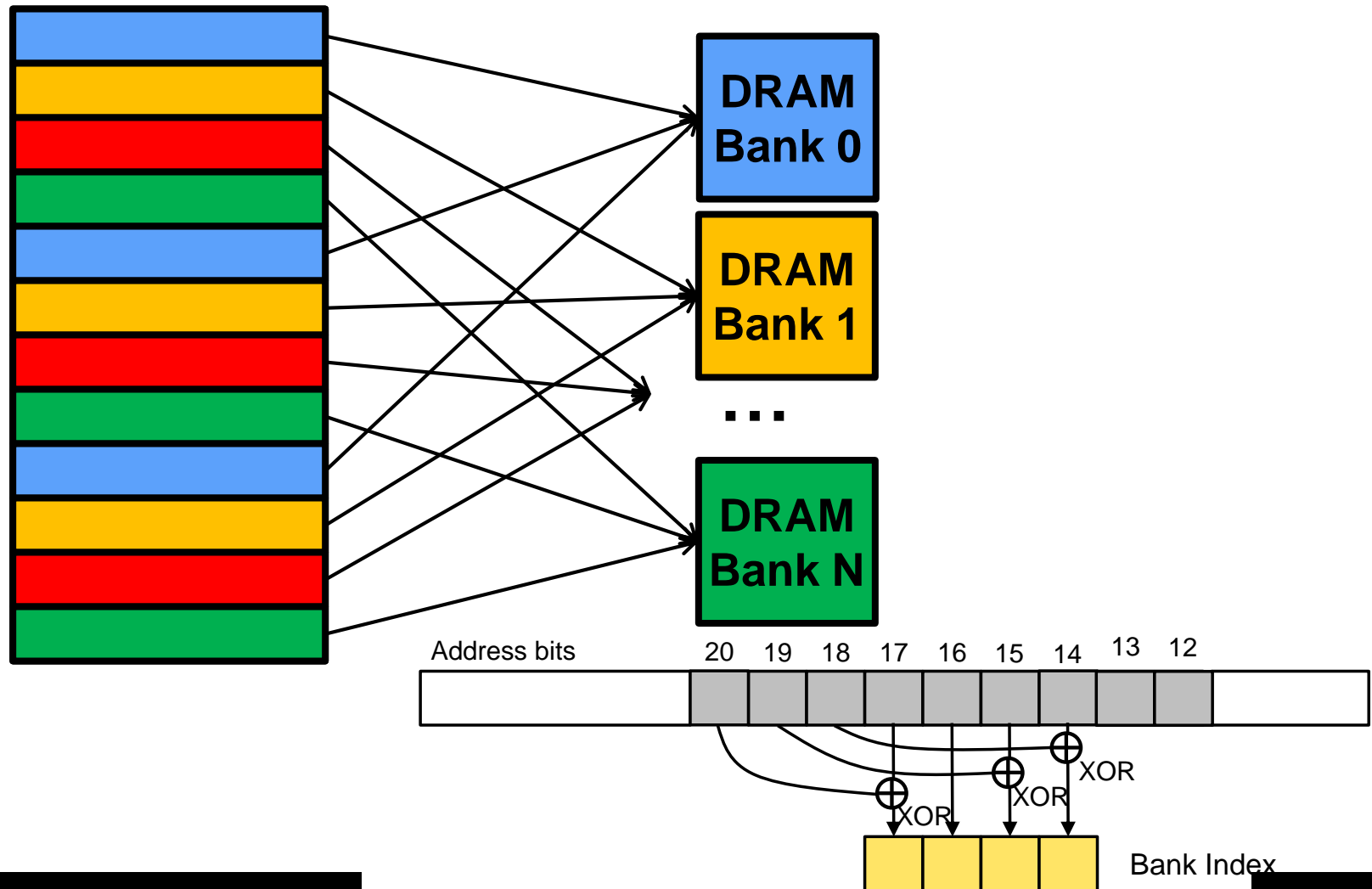


Cache Partitioning (Coloring)

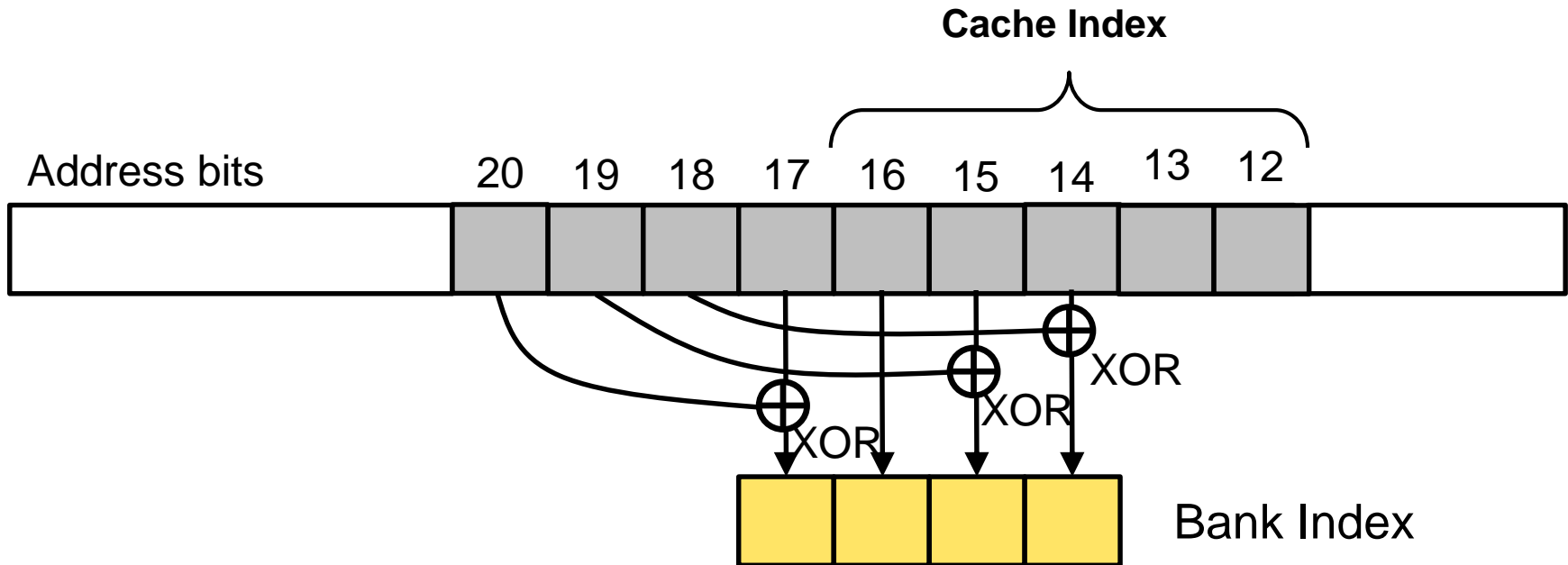


Bank Partitioning (Coloring)

Main Mem



Cache and Bank Address Bits



E.g. 2 bank bits
2 cache bits
1 shared bit

		Bank			
		00	01	10	11
Cache	00	X		X	
	01	X		X	
	10		X		X
	11		X		X

Row-Bank Address Bit Xoring Improves Coverage

If two additional bits are xor with bank bits we can get all combinations

Bank Colors												
	row	bank	= 00	row	bank	= 01	row	bank	= 10	row	bank	= 11
	00	00		01	00		10	00		11	00	
	01	01		00	01		11	01		10	01	
	10	10		11	10		00	10		01	10	
	11	11		10	11		01	11		00	11	
	00	X		X	X		X					
Cache Colors	01	X	X	X	X							
	10	X	X	X	X							
	11	X	X	X	X							

Coordinated Cache and Bank Partitioning & Core Allocation

Avoid conflicting color assignments

Take advantage of different conflict behaviors

- Banks can be shared within same core but not across cores
- Cache cannot be shared within or across cores
- Coordinated core and bank color allocation

Take advantage of sensitivity of execution time to cache

- Task with highest sensitivity to cache is assigned more cache
- Diminishing returns taken into account

Two algorithms explored

- Mixed-Integer Linear Programming



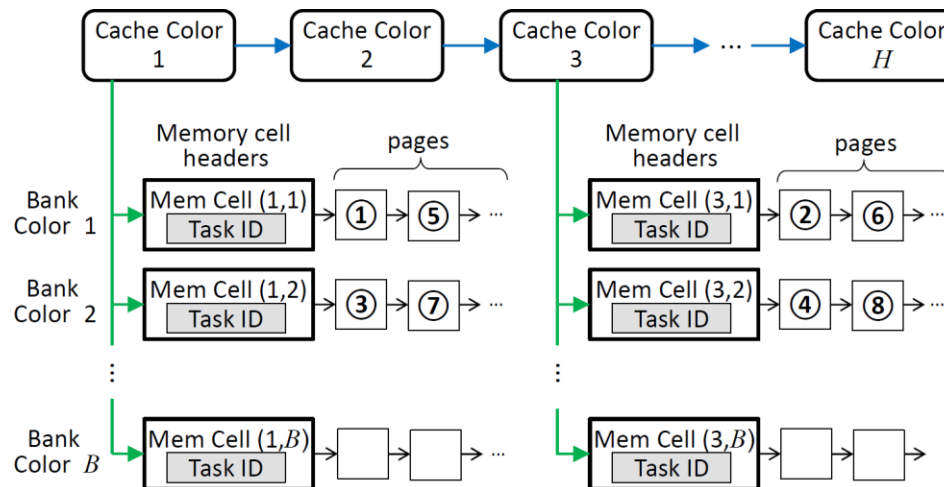
Implementation of Cache+Bank Coloring

Linux / RK : Kernel Memory Manager

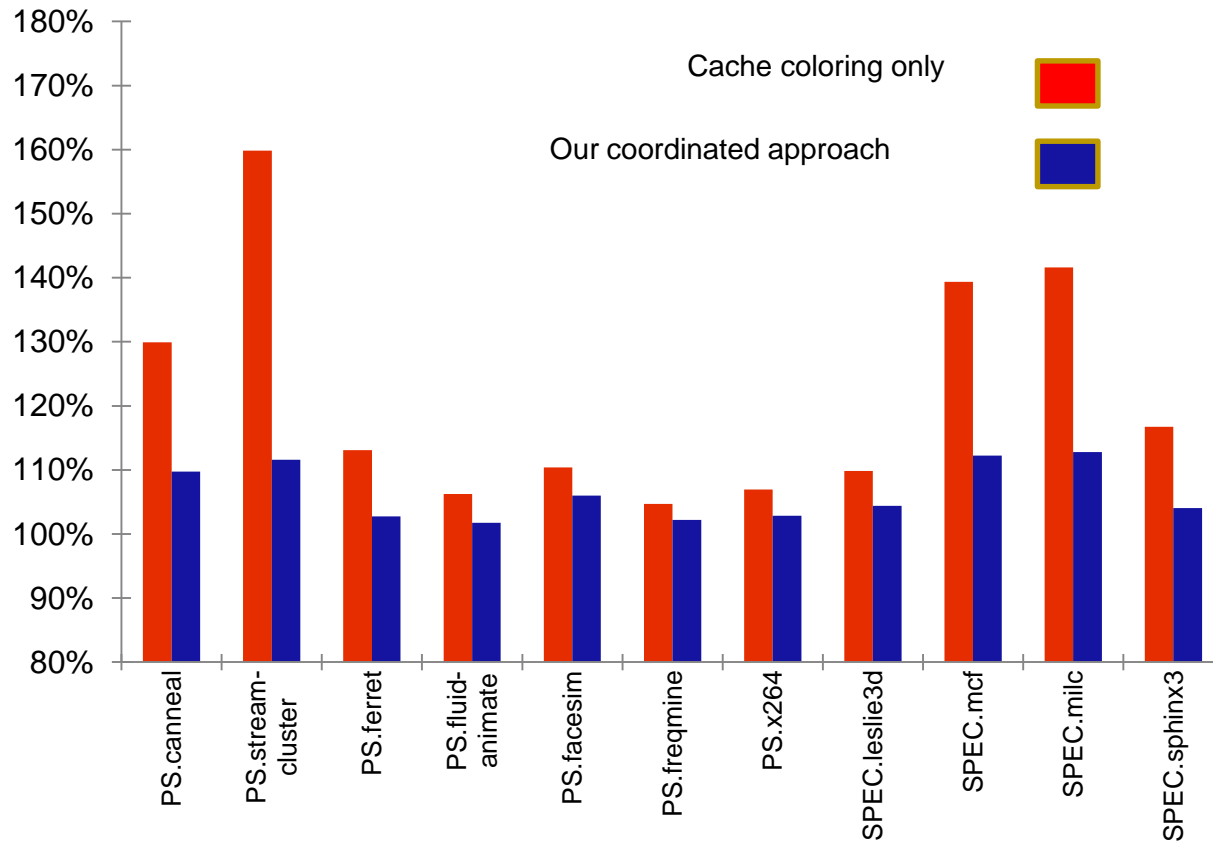
Memory reserves with set of bank and cache colors

Pages are classified in cache and bank colors

Added to resource sets that are attached to multiple processes/threads



Experimental Results



Limited Number of Private Partitions

Private partitions significantly reduces usable memory

- Number of bank/cache cells in memory
 - Number of cells = $(B*H)$. Size of cell $C = \frac{M}{(B*H)}$
 - With: M = size of memory, B = # bank colors, H = # cache colors
 - E.g. Intel core i7 2600
 - $M = 4GB$, $B = 16$, $H = 32$ $C = \frac{4GB}{16*32} = 8MB$
- Private partitions \equiv one cell per cache color & one cell per bank color
 - Number of private partitions $PP = \min(B, H)$
 - E.g. Intel core i7 2600 : $PP = \min(16, 32) = 16$
- Extreme (using all private partitions) total usable private partition memory



Allowing Sharing

In Partitioned Scheduling OK to share banks within core

- Number of banks are no longer a restriction: $PP = H$
- Partitions sharing banks in a core
 - # Sets of independent partitions $I = N$; N = number of cores
 - Memory utilization (uniform partitions) = $\frac{M}{I}$
- Intel Core i7 2600: $I = N = 4$
 - Memory utilization (uniform partitions) = $\frac{4GB}{4} = 1GB = 25\%$

Need better utilization

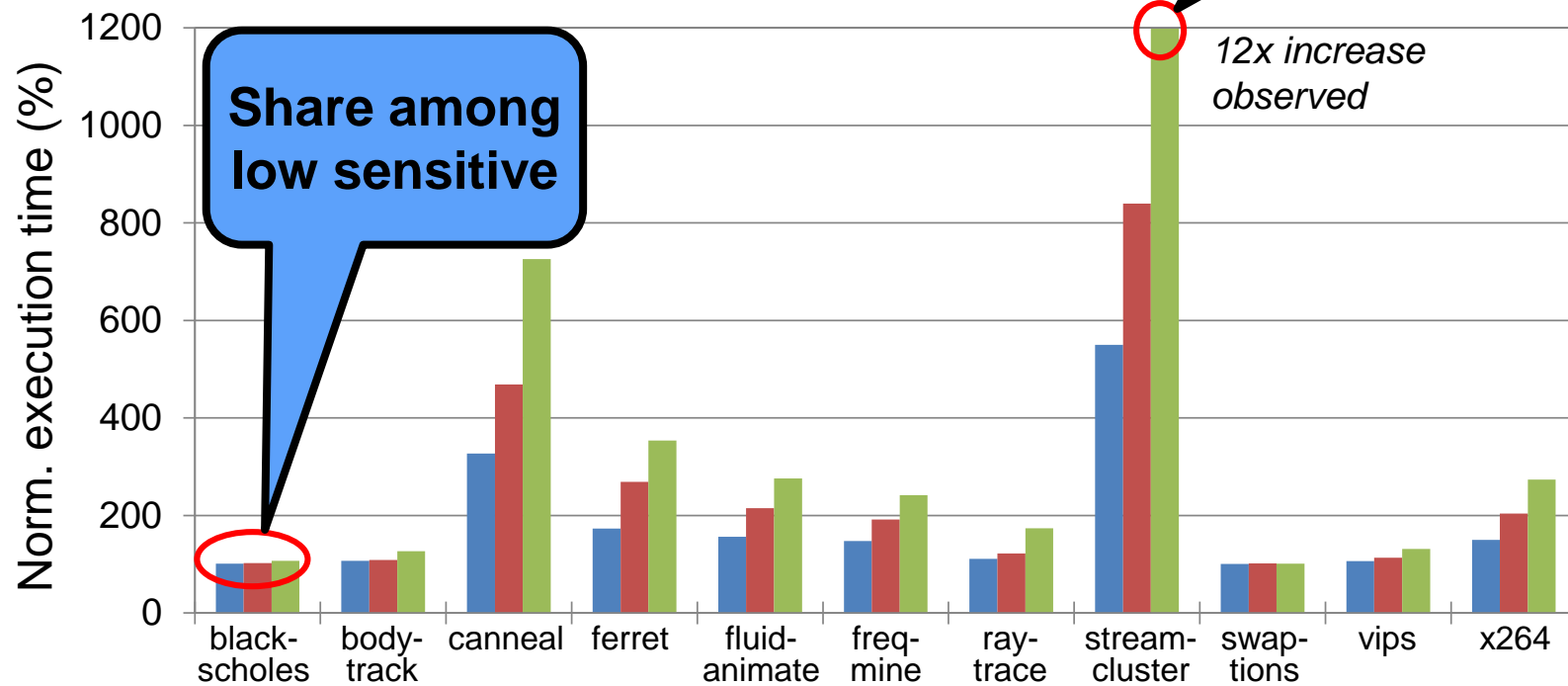
Partitions may not be enough for number of tasks

Predictable Sharing

Exploit different sensitivity

Bounding interference

Policing and enforcement



Bank Partitioning (Coloring) + Timing Analysis

Explicitly considers the timing characteristics of major DRAM resources

- Rank/bank/bus timing constraints (JEDEC standard)
- Request re-ordering effect

Bounding memory interference delay for a task

- Combines request-driven and job-driven approaches
- | | |
|----------------------------|--|
| Task's own memory requests | Interfering memory requests during the job execution |
|----------------------------|--|

Software DRAM bank partitioning awareness

- Analyzes the effect of dedicated and shared DRAM banks

Response-Time Test

- **Memory interference delay cannot exceed any results from the RD and JD approaches**
 - We take the smaller result from the two approaches
- **Extended response-time test**

$$\begin{aligned}
 & \boxed{R_i^{k+1} = C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil \cdot C_j} \quad \text{Classical iterative response-time test} \\
 & + \min \left\{ \boxed{H_i \cdot RD_p + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil \cdot H_j \cdot RD_p}, \boxed{JD_p(R_i^k)} \right\} \\
 & \qquad \qquad \text{Request-Driven (RD) Approach} \qquad \qquad \text{Job-Driven (JD) Approach}
 \end{aligned}$$

Memory-Interference Aware Task Allocation

Observations

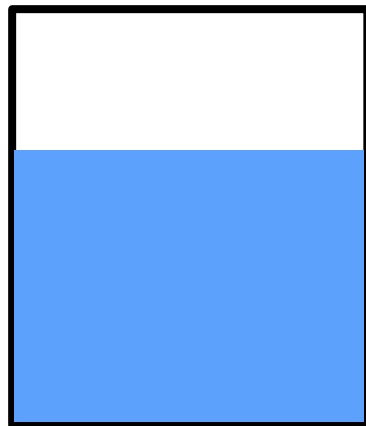
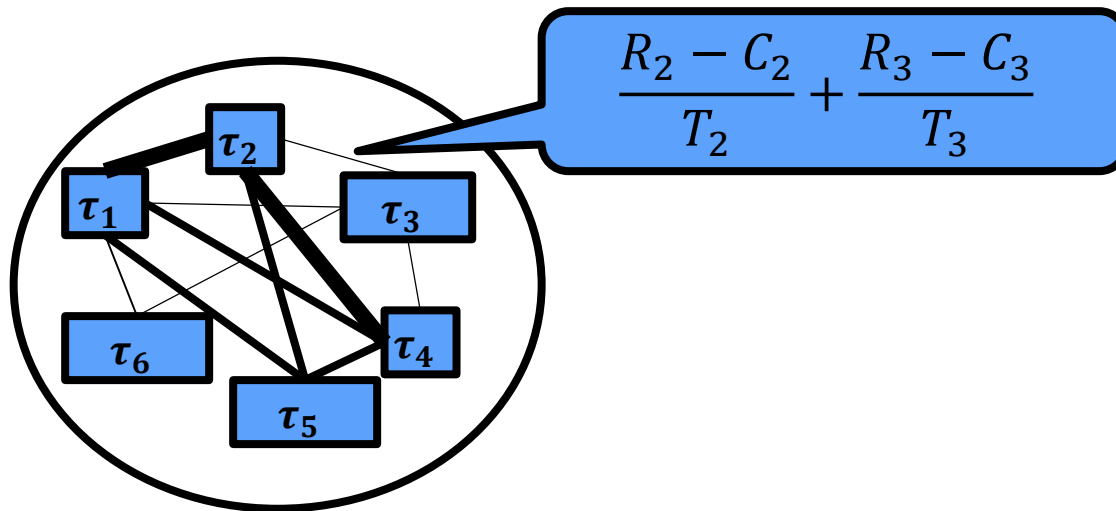
- Memory interference due to tasks running in other cores
- Tasks running on same core do not interfere with each other
- Collocate memory-intensive tasks on same core

Graph $G = (V_i, E_{i,j})$: $V_i = \tau_i$, $E_{i,j} = interference(\tau_i, \tau_j)$,
 $weight(E_{i,j}) = \frac{R_i - C_i}{T_i} + \frac{R_j - C_j}{T_j}$

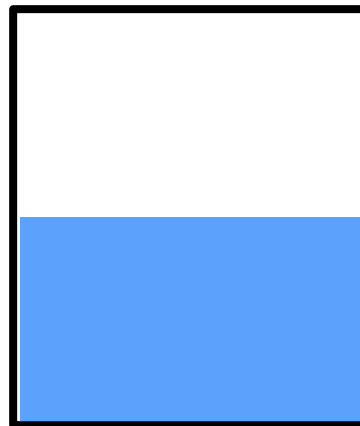
Following BFD:

1. Try to deploy first un-deployed subgraph on bin (core)
2. If cannot
 - break graph with minimum cut (minimize edge weights)
 - One piece that fits largest gap + rest
3. Add to undeployed subgraphs

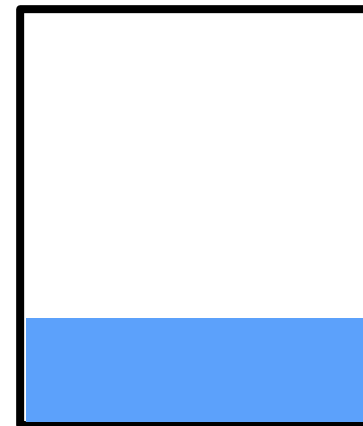
Minimum-Cut Memory Interference Packing



Core 1

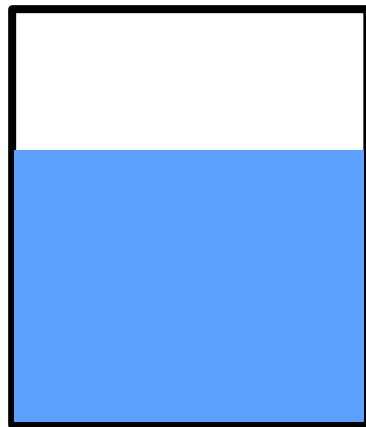
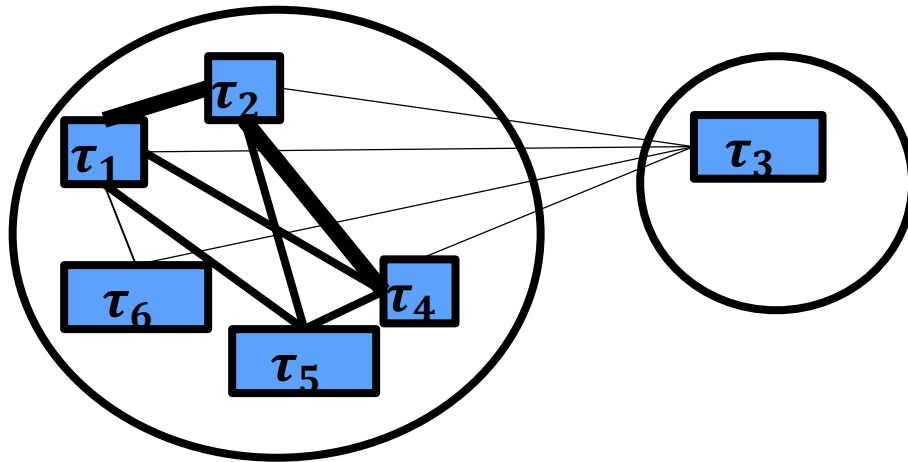


Core 2

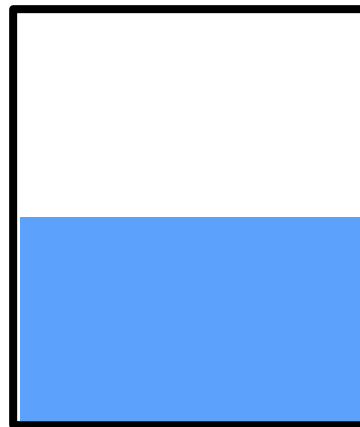


Core 3

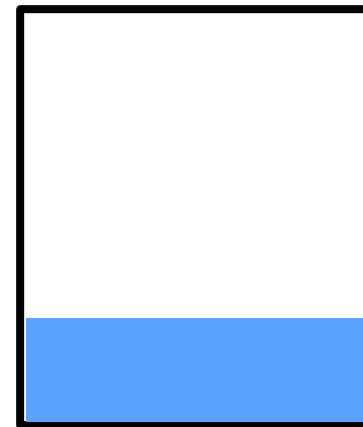
Minimum-Cut Memory Interference Packing



Core 1

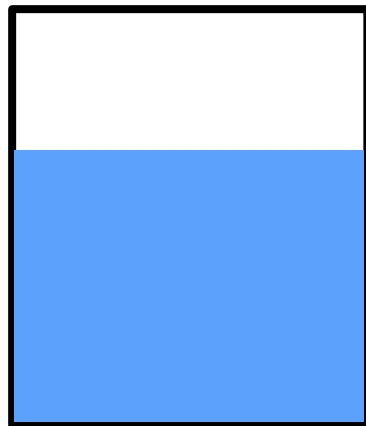
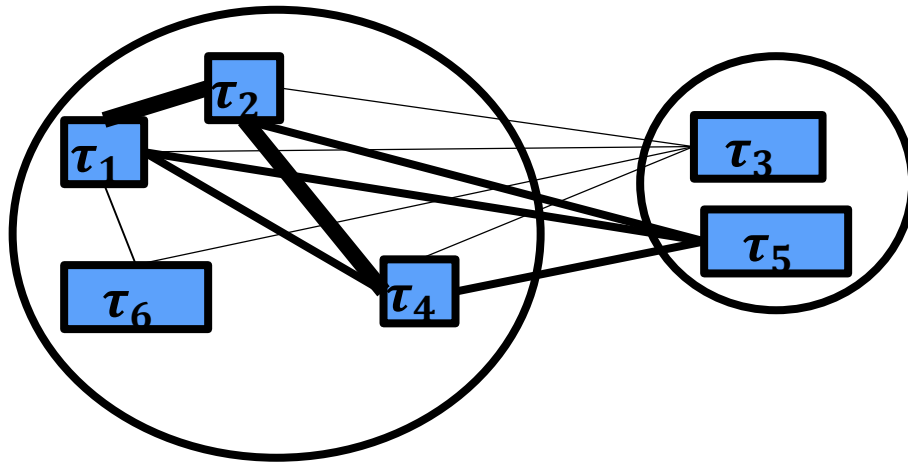


Core 2

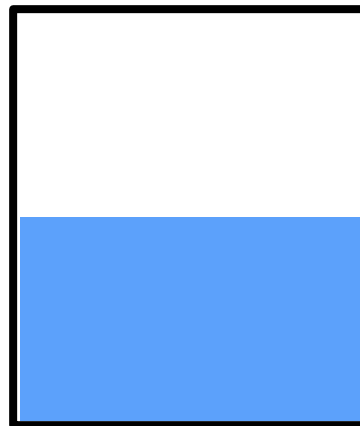


Core 3

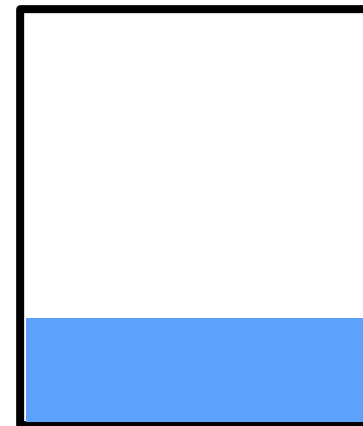
Minimum-Cut Memory Interference Packing



Core 1

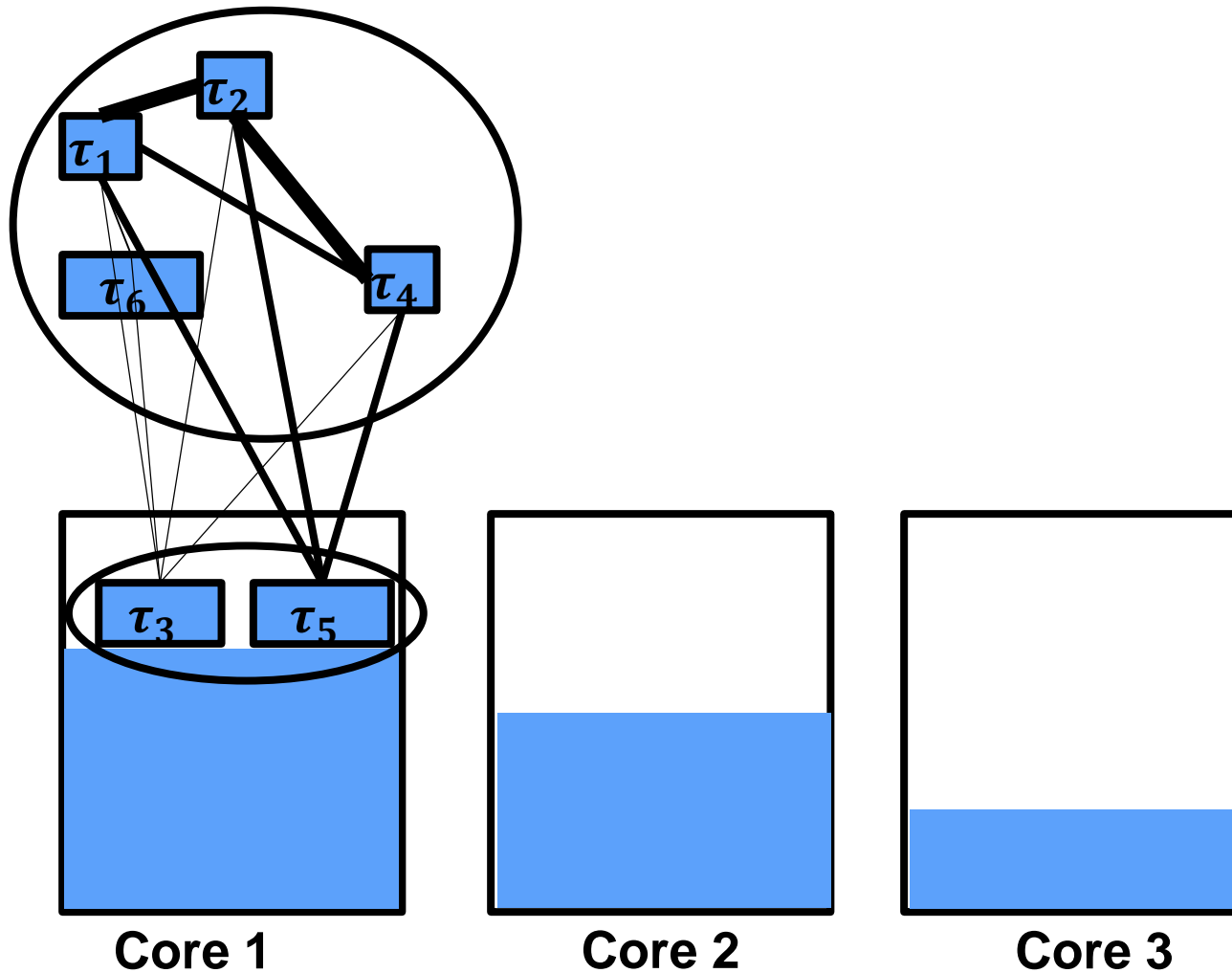


Core 2

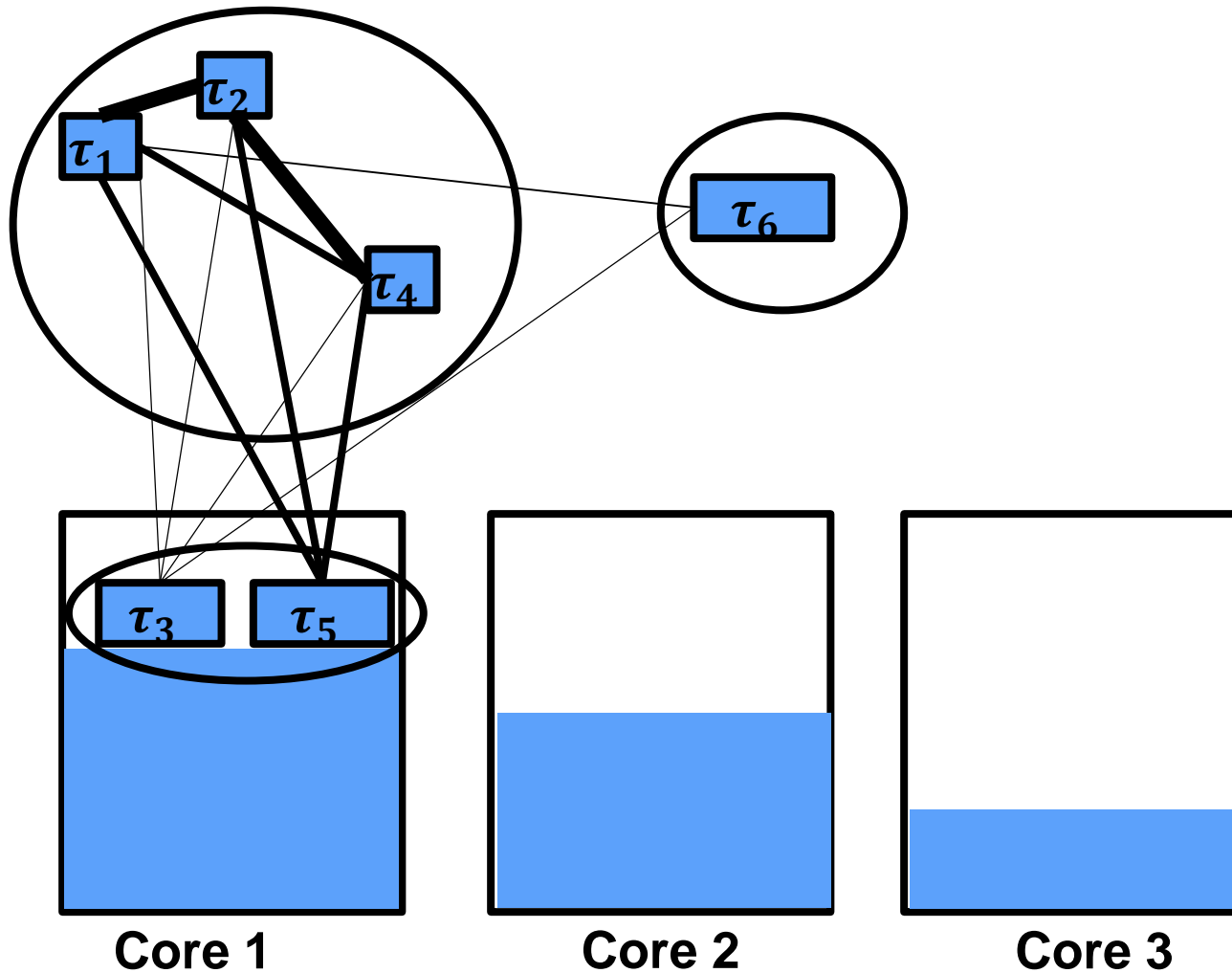


Core 3

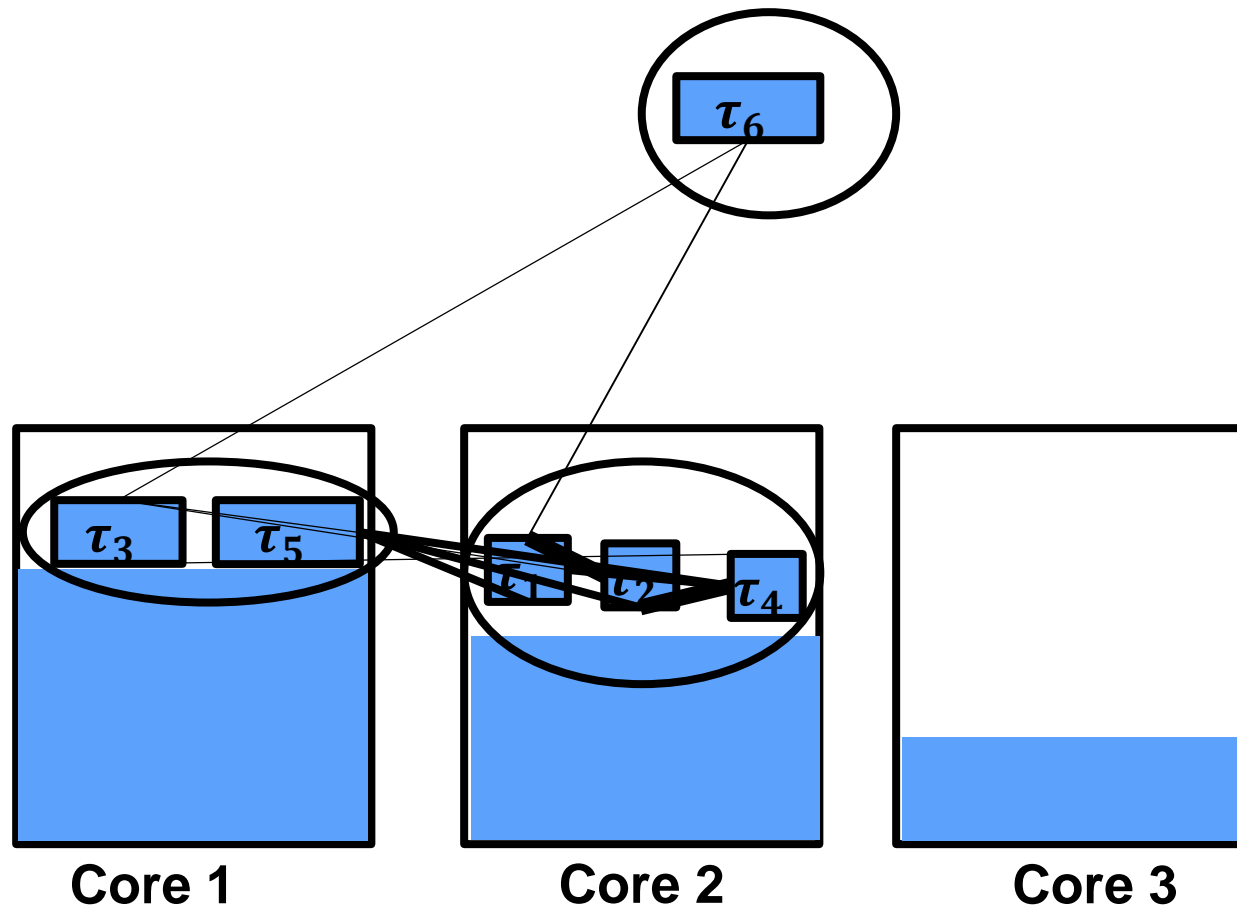
Minimum-Cut Memory Interference Packing



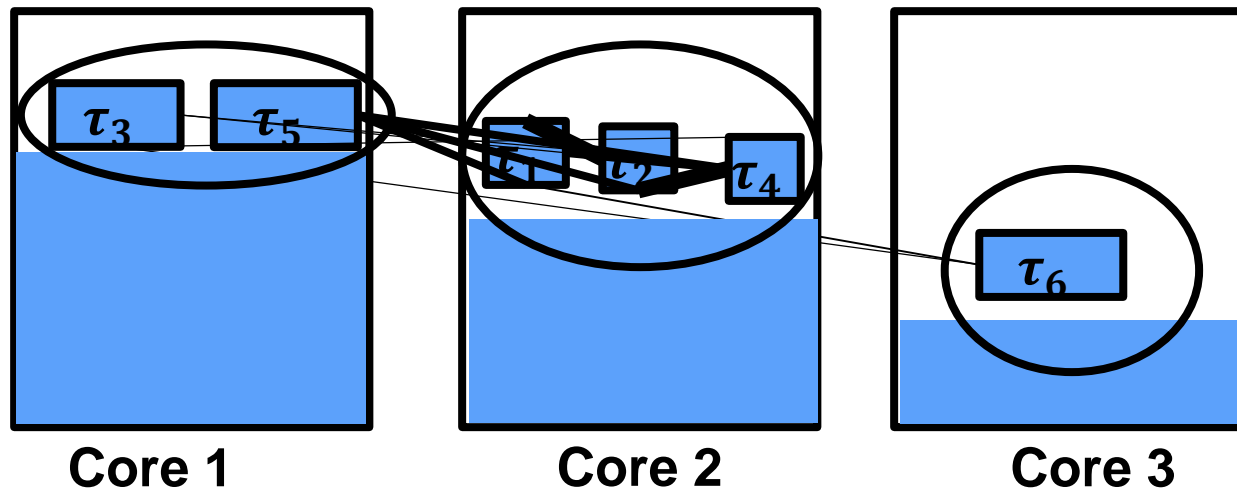
Minimum-Cut Memory Interference Packing



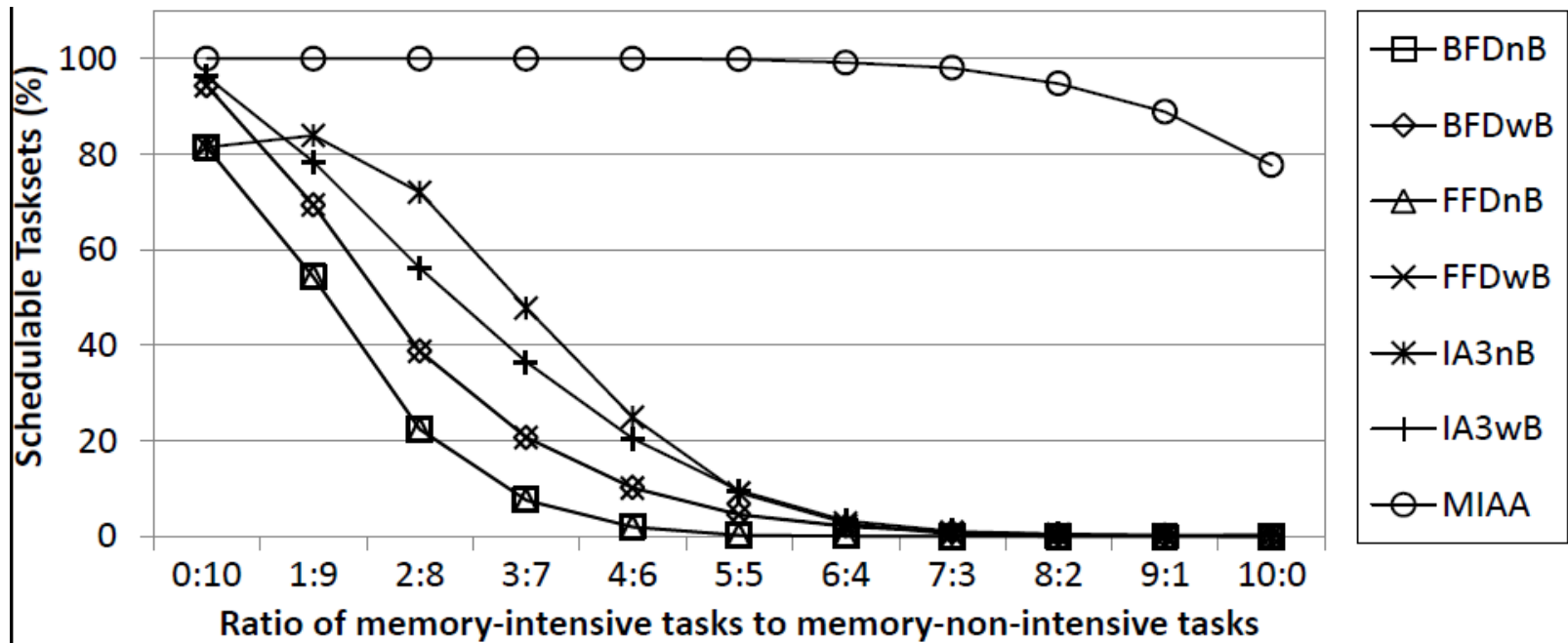
Minimum-Cut Memory Interference Packing



Minimum-Cut Memory Interference Packing



Memory-Interference Aware Task Allocation (MIAA)

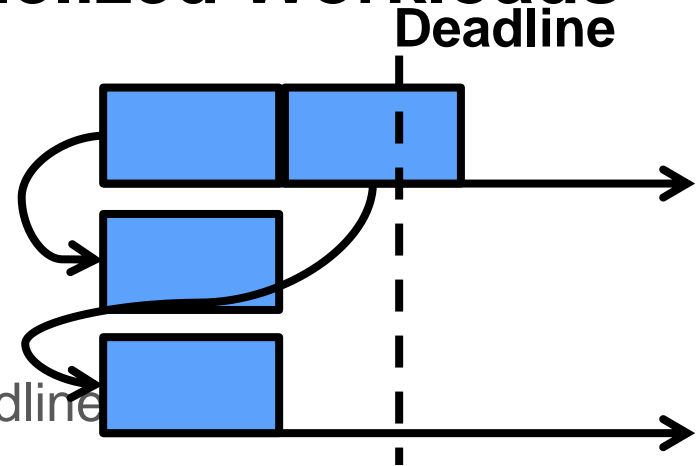


IA³ M. Paolieri, E. Quiñones, F. Cazorla, R. Davis, and M. Valero. IA3: An interference aware allocation algorithm for multicore hard real-time systems. RTAS 2011.

Resource Conflicts for Parallelized Workloads

Parallelization

- Computation time > Deadline
 - Must parallelized to meet deadline
 - Guarantee always finish before deadline



Resource interference within a task

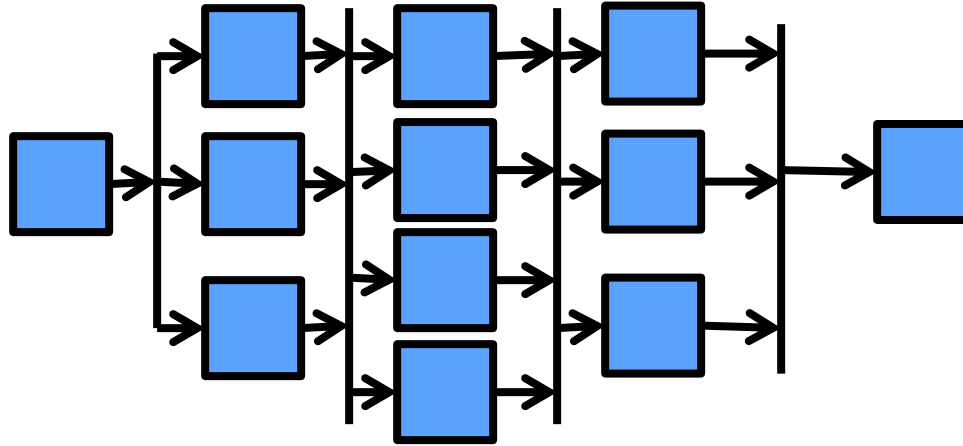
- Due to parallel subtasks
- Need to share memory to communicate

Predictable sharing

- Compatible with efficient parallelized task schedulers

Parallelized Task Scheduling

Developed a staged execution model



Scheduled under Global Earliest-Deadline First

- Most efficient scheduling for staged execution
 - If task schedulable under optimal scheduler our scheduler need at most twice the speed to schedule task

Challenges for Parallelized Task Resource Management

Intra-task partitions

- Threads with different sensitivities
- Assign different partitions to different parts of same tasks
 - Down to different colors for each page of a task

Inter-task shared partitions

- Shared partitions between parts of different tasks

Intra-task memory bus interference



Hardware and Software Profiling

Hardware

- Mapping of memory bits for cache and bank index
- Randomization strategies

Software

- Bound on number of memory accesses
- Temporal and spatial locality of accesses
- Techniques
 - Model checking (better term?)
 - Variable placement and access
 - Control-flow-based temporal and spatial locality
 - Profiling
 - Performance counters
 - Valgrind



Contact Information

Bjorn Andersson

Senior Member of Technical Staff

Telephone: +1 412.268.9243

Email: baandersson@sei.cmu.edu

